Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Kristijan Mirčeta

Ekstrakcija časovnega znanja iz dogodkov v spletnih novicah

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Lovro Šubelj

Ljubljana, 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Never Ending Language Learner je sistem Univerze Carnegie Mellon, ki se z branjem spleta uči različnih konceptov in relacij med njimi. Pri tem pa se ne zaveda, da so nekatere izmed relacij časovne kar pomeni, da so veljavne le za določene trenutke v času. Možno rešitev predstavlja sistem EventRegistry Instituta Jožef Stefan, ki v realnem času zbira spletne novice in v njih prepoznava dogodke. Le-ti pa lahko nosijo informacijo o spremembah veljavnosti časovnih relacij.

V diplomskem delu razvijte sistem za avtomatsko odkrivanje dogodkov, ki spremenijo časovne relacije. Za to uporabite pristope strojnega učenja in odkrivanja znanj iz besedil ter metode aktivnega učenja za pohitritev ročnega označevanja dogodkov. Delovanje razvitega sistema preizkusite na primeru izbrane časovne relacije.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Kristijan Mirčeta sem avtor diplomskega dela z naslovom:

Ekstrakcija časovnega znanja iz dogodkov v spletnih novicah

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Lovra Šublja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 14. septembra 2015 Podpis avtorja:

Najprej se zahvaljujem svojemu mentorju doc. dr. Lovru Šublju za vso strokovno pomoč in nasvete, ki mi jih je nudil tokom nastanka diplomskega dela.

Hvala kolegom Gregorju Lebanu, Jenyi Belyaevi in posebej mentorju na IJS Aljažu Košmerlju za vso pomoč in pri diplomi in v sodelovanju na IJS. Hvala Marku Grobelniku za ponujeno priložnost in kolegom na CMU za konstruktivno in prijetno sestankovanje glede diplomskega dela. Za strokovno pomoč in moralno podporo se zahvaljujem tudi mentorici na IJS Maji Šrkjanc, ki mi je omogočila da sem se v okolici IJS počutil kot doma.

Posebno za prijateljstvo se zahvaljujem prijateljema Bojanu in Vučku.

Najlepša zahvala pa gre babici Oliveri, mami Marici in očetu Bojanu, za ljubezen, ki so mi jo izkazovali ob vseh vzponih in padcih. Najlepša hvala pokojnemu dedku Petru, katerega, v mojih očeh, veliki znanstveni uspehi navdihujejo in ženejo tudi mene.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CA	Classification Accuracy	klasifikacijska točnost
AUC	Area Under ROC Curve	površina pod ROC krivuljo
ROC	Receiver Operating Characteristic	sprejemna operativna karakteristika
SVM	Support Vector Machine	metoda podpornih vektorjev
TF-IDF	Term Frequency times Inverse Document Frequency	frekvenca besede v dokumentu krat relativna redkost besede v korpusu
CMU	Carnegie Mellon University	univerza Carnegie Mellon
NELL	Never Ending Language Learner	Večno učeči se učenec jezika
ER	EventRegistry	registrar dogodkov

Povzetek

V diplomskem delu predstavimo sistem Never Ending Language Learner, v nadaljevanju NELL, ki z branjem spleta gradi bazo znanja v obliki konceptov in relacij med njimi. Nekatere relacije so odvisne od časa kar pomeni, da je njihova vrednost lahko drugačna ob dveh trenutkih v času. Takšnim relacijam pravimo časovne relacije. Le-te se naprej delijo na relacije, ki se zgolj zgodijo in na relacije, ki se začnejo in končajo, oziroma časovne relacije z enim kritičnim trenutkom v času in časovne relacije z dvema kritičnima trenutkoma. Kritični trenutek je trenutek ob katerem se vrednost relacije spremeni. Sprememba je lahko začetek, ki je prehod vrednosti iz 0 v 1, konec, ki je prehod vrednosti iz 1 v 0 ali dogodek, ki vrednost spremeni iz 0 v 1 in za tem nazaj iz 1 v 0. Relacije z dvema kritičnima trenutkoma imajo začetek in konec, relacije z enim kritičnim trenutkom pa le dogodek.

Sistem NELL ima problem s prepoznavanjem kritičnih trenutkov za relacije kar pomeni, da ne ve kdaj se je neka relacija zgodila oziroma začela ali končala. Splošen problem časovnih relacij je kako pridobiti za relacijo metapodatke o tem kdaj se je zgodila, za relacije z enim kritičnim trenutkom oziroma kdaj se je začela in končala za relacije z dvema trenutkoma.

V diplomskem delu se ukvarjamo s specifičnim podproblemom problema časovnih relacij, kako najti besedila, ki vsebujejo informacije o kritičnih trenutkih. Predstavimo sistem EventRegistry, ki nabira časopisne članke iz različnih virov in jih grupira v dogodke, ki jih ponuja kot podatke, ki opisujejo karkoli značilnega kar se je zgodilo. Nekateri od teh dogodkov vsebujejo informacije o kritičnih trenutkih.

Predlagamo splošen sistem za odkrivanje dogodkov, ki vsebujejo informacijo o kritičnih trenutkih za relacije z dvema. Sistem deluje na podlagi klasifikacijskih algoritmov, ki z uvrščanjem ločijo dogodke z informacijo o kritičnih trenutkih od drugih. Ker klasifikacijski algoritmi zahtevajo označene podatke, označevanje podatkov pa je izjemno drago in zamudno delo, predlagan sistem nadgradimo še s strategijami aktivnega učenja, ki poskušajo zmanjšati ceno označevanja podatkov. Razvit sistem simuliramo in analiziramo na primeru časovne relacije ImaZakonca(x,y) in poročamo o njegovi uspešnosti. Za konkretno relacijo se izkaže, da je problem dobro rešljiv, saj v klasifikaciji dosežemo AUC blizu 0.90.

Ker podatke označimo tako, da z njimi na enostaven način lahko odkrijemo tudi tip kritičnega trenutka v času, ki ga dogodek vsebuje, predstavimo rezultate tudi za ta podproblem, na primeru konkretne relacije ImaZakonca(x,y). Tudi za ta podproblem se izkaže, da je s klasifikacijo dobro rešljiv, saj prav tako dosežemo AUC blizu 0.90.

Ključne besede: strojno učenje, odkrivanje znanj v besedilih, aktivno učenje, podatkovno rudarjenje.

Abstract

In this thesis we describe the system Never Ending Language Learner referred to as NELL that builds a knowledge base in the form of concepts connected by relations, by reading the web. Some relations are dependent on time, which means that their value may be different at two moments in time. We call them temporal relations. These are further divided into relations that happen and relations that start and end or equivalently, relations with one critical moment in time and relations with two critical moments. A critical moment is a moment at which the value of the relation changes. The change may be the beginning, which is the transition from 0 to 1, the ending, which is the transition from 1 to 0, or the event, which changes the value of the relation from 0 to 1 and then quickly back from 1 to 0. Relations with two critical moments in time have a beginning and an end, whereas relations with one such moment only have a happening.

The system NELL has a problem with the recognition of such critical moments for relations, which means that it doesn't know when some relation began or ended, or in the case of relations with one critical moment, happened. The general problem of temporal relations asks how to get metadata for a relation, about when it happened for relations with one critical moment in time, and when it began and ended for the relations with two.

In the thesis we address the specific subproblem of the problem of temporal relations that asks how to find text that contains information about critical moments. We describe the system EventRegistry, which collects news paper articles from various sources and groups them into events, which are

represented as data about various significant happenings. Some of these events contain information about critical moments in time.

We propose a general system for detecting events, which contain information about critical moments for relations with two of them. The system is based on classification algorithms, which, by classification, separate the events that contain information about critical moments from the others. Because classification algorithms demand labeled data, and labeling is extremely costly and slow, we improve the proposed system with active learning strategies, which try to reduce the cost of labeling data. We simulate and analyze the proposed system for the case of the relation HasSpouse(x,y) and report the success of it's performance. For this concrete relation it turns out that the problem is very solvable, as we report AUC near 0.90 for the classification.

Because the data is labeled in a way that allows us to also detect the type of critical moment contained in the event in a simple way, we present the results for this subproblem as well, for the concrete relation HasSpouse(x,y). This problem also turns out to be highly solvable by classification, as we also achieve AUC near 0.90.

Keywords: machine learning, text mining, active learning, data mining.

Poglavje 1

Uvod

Strojno učenje je področje umetne inteligence, ki omogoča računalniškim programom, da se učijo brez, da bi bili eksplicitno programirani [1]. Pri klasičnem problemu strojnega učenja imamo navadno na voljo množico podatkov, iz katerih želimo ekstrahirati neko znanje. Eden najpogostejših takšnih problemov je tako imenovana klasifikacija. Cilj klasifikacije je program naučiti avtomatske delitve podatkov v razrede oziroma, ekvivalentno, podatkom dodeliti oznake. Za to imamo na voljo začetno množico podatkov katere elementom pravimo primeri. Vsak primer ima atributni prostor in razred, za vsak element atributnega prostora pa je podana vrednost. Tako se na podlagi vrednosti primerov v atributnem prostoru učimo razlikovati med različnimi razredi. Na tak način lahko za nove primere, za katere ne poznamo razreda, na podlagi vrednosti atributov razred avtomatsko napovemo.

Enostaven primer takšnega tipa strojnega učenja je filtriranje neželene elektronske pošte. Vsak od nas je že imel problem s tem, da je v svoj elektronski poštni nabiralnik prejel elektronsko pošto, ki je ni želel. Na podlagi že dobljene neželene elektronske pošte, lahko program primerja le-tega z elektronsko pošto, ki ni neželena in se tako nauči avtomatsko razločevati med njima. Posledica je, da program neželeno elektronsko pošto še preden jo vidimo v nabilralniku, avtomatsko vrže v koš. To je analogno na to, da se je program naučil razlikovati med dvema razredoma, konkretno razredom

neželene elektronske pošte in takšne pošte, ki ni nezaželena.

Omenjen program ima funkcijo tega, da razbremeni človeka s tem, da opravlja njegovo nalogo. Na nek način bi lahko rekli, da je program ali agent inteligenten. Če ga primerjamo s človekom pa hitro opazimo, da se tak program in človek močno razlikujeta. Človek se je za razliko od programa sposoben naučiti veliko različnih tipov znanja z leti različnih izkušenj ter s trenutnim znanjem spodubuditi učenje novih oblik znanja. Če bi želeli zgraditi sistem, ki se je sposoben učiti tako, kot se uči človek, bi morali problem preoblikovati.

Različne raziskave so naslovile problem gradnje agentov strojnega učenja, ki se učijo čez dolge časovne intervale, kot na primer "life long learning" [2], "learn to learn" [3]. Le redki sistemi pa demonstrirajo tak stil učenja v praksi. Obstajajo arhitekture, katerih cilj je generalno reševanje problemov in učenje, kot na primer "SOAR" [4], "ICARUS" [5], "PRODIGY" [6]. Nobeden od omenjenih sistemov pa ni kontinuirano deloval dalj časa. Po našem vedenju, obstaja v času nastanka te diplome le en tak sistem. Never Ending Language Learner NELL je projekt, ki ga razvijajo na univerzi Carnegie Mellon in kontinuirano teče od januarja 2010 [7]. NELL je v najbolj grobem smislu poskus simulacije paradigme po kateri se učijo ljudje. Od svojega začetka se je naučil že 80 milijonov verjetij, kot je na primer seServiraSkupajZ(Čaj, Piškoti) ali jePredsednikDržave(Barack Obama, ZDA). S tem znanjem lahko NELL odgovarja na poizvedbe kot je na primer "Kdo je predsednik ZDA?". NELL se uči z branjem velikih količin spletnih strani in s časom izboljšuje natančnost ekstrahiranja verjetij iz besedil, ki jih "prebere". V poglavju 2 podrobneje predstavimo sistem NELL in njegov formalno definiran problem. NELL bi lahko primerjali z Googlovim spletnim iskalnikom. Če bi Googlov spletni iskalnik vprašali "Kdo je predsednik ZDA?", bi nam ta odgovoril s seznamom spletnih strani, urejenih po verjetju, da se na njih skriva podatek, ki ga želimo. NELL bi nam na vprašanje odgovoril z natančnim odgovorom, ki ga iščemo, če bi ga poznal. Po drugi strani pa ima NELL problem z določenim tipom poizvedb. Iz svojih virov se uči o dejstvih, dočim ne ve kdaj

so aktualna. S tem mislimo, da je neka binarna relacija v nekem trenutku v času lahko veljala, v naslednjem trenutku pa je prenehala veljati. Konkretna instanca relacije JePredsednikDržave za katero velja slednje je na primer JePredsednikDržave(George W. Bush, ZDA). Na poizvedbo "Kdo je bil leta X predsednik ZDA", bi odgovoril z množico vseh predsednikov v zgodovini ZDA.

Torej spoznamo, da obstajajo relacije, katerih veljavnost je odvisna od časa in, da NELL nima občutka za čas. V diplomskem delu predstavimo sistem EventRegistry [8], ki je sposoben NELL-u pomagati k rešitvi problema časovno odvisnih relacij in predlagamo korak k rešitvi, ki sklopi oba sistema, oziroma jima omogoči komunikacijo. Slednje dosežemo z razvojem splošnega sistema, ki izkoristi podatke, ki jih ponuja EventRegistry za iskanje kritičnih trenutkov v času za različne časovne relacije. Kritični trenutki v času so trenutki v času ob katerih se stanje neke relacije spremeni. EventRegistry ponuja podatke v obliki dogodkov, ki opisujejo karkoli značilnega kar se je zgodilo, nekateri od teh dogodkov pa vsebujejo informacijo o kritičnih trenutkih v času. Razviti splošni sistem z metodami klasifikacije najde omenjene dogodke in v njih prepozna tip kritičnega trenutka v času, ki ga dogodek vsebuje. Tu je tip trenutka lahko začetni, ki vrednost neke instance relacije spremeni iz 0 v 1, končni, ki to vrednost spremeni iz 1 v 0, ali dogodkovni, ki pove, da se je nekaj zgodilo in tako spremeni vrednost iz 0 v 1, a takoj za tem nazaj v 0. Podrobno razvit sistem opišemo v poglavju 3.

Poglavje 2

Metode in tehnologije

2.1 Strojno učenje

Strojno učenje je področje umetne inteligence, ki omogoča računalniškim programom, da se učijo brez, da bi bili eksplicitno programirani [1]. Poznamo dve vrsti strojnega učenja: nadzorovano učenje in nenadzorovano učenje. V diplomskem delu se ukvarjamo z nadzorovanim učenjem.
Problem nadzorovanega učenja je zastavljen na sledeč način:

Na voljo imamo množico podatkov $D_L := \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), ..., (\mathbf{x_m}, y_m)\}$, velikosti $|D_L| = m$, katere vsak element je sestavljen iz vrednosti iz prostora atributov $\mathbb{X} \subseteq \mathbb{R}^n$, ki je n dimenzionalen in prostora oznak Y, ki je skalaren. Cilj nadzorovanega učenja je poiskati funkcijo h, ki najbolje aproksimira preslikavo f, ki vrednosti v atributnem prostoru $\mathbb{X}$ preslika v Y.

$$h \colon \mathbb{X} \mapsto Y \tag{2.1}$$

Razlikujemo dve vrsti nadzorovanega učenja, ki se ločita glede na vrsto prostora Y:

- uvrščanje ali klasifikacija - množica Y je končna množica diskretnih vrednosti.
- regresija - $Y \subset \mathbb{R}$, torej je v Y neomejeno realnih vrednosti.

Funkcijo h, ki ji pravimo tudi hipoteza, iščemo v tako imenovanem hipoteznem prostoru $\mathbb{H}$, ki je navadno prevelik za izčrpno preiskovanje. Za preiskovanje hipoteznega prostora so bili razviti algoritmi strojnega učenja, ki $\mathbb{H}$ preiskujejo učinkovito.

2.2 Klasifikacijski algoritmi

V tem poglavju predstavimo različne pristope za preiskovanje hipoteznega prostora $\mathbb{H}$ oziroma algoritme strojnega učenja. Ker v diplomskem delu uporabljamo le klasifikacijske algoritme, predstavimo tri primere le-teh.

2.2.1 Naivni Bayesov klasifikator

Naivni Bayesovi klasifikatorji, ki temeljijo na popularnem Bayesovem verjetnostnem izreku, so družina linearnih klasifikatorjev in so znani po svoji enostavnosti in učinkovitosti [10]. Tako kot vsak klasifikacijski algoritem, se učijo na množici podatkov, katere vsak element ali primer, je sestavljen iz vrednosti v prostoru atributov in razreda. Temeljijo na predpostavki, da so atributi neodvisni en od drugega, od tu pa izhaja ime "naivni". Glavni razlog za predpostavko o neodvisnosti je enostavnost.

Čeprav spremenljivke niso neodvisne, pa se klasifikator v praksi pogosto dobro izkaže kljub temu, da operira pod nerealistično predpostavko. Lahko bi rekli, da klasifikator deluje dobro, ko so spremenljivke **dovolj** neodvisne. Temelj naivnega Bayesovega klasifikatorja, Bayesov verjetnostni izrek pravi sledeče:

$$P(y_j|x_i) = \frac{P(x_i|y_j)P(y_j)}{P(x_i)}, \tag{2.2}$$

Kjer je x_i instanca prostora atributov, y_j pa konkreten razred. Verjetnosti $P(y_j|x_i)$ pravimo tudi posteriorna verjetnost. Zgornji izraz bi lahko drugače opisali tudi na sledeč način:

$$\text{posteriorna verjetnost} = \frac{\text{razredno pogojna verjetnost} \times \text{apriorna verjetnost}}{\text{dokaz}}. \tag{2.3}$$

Naivni Bayes za nek nov primerek x_i izbere razred y_j, ki maksimizira posteriorno verjetnost

$$\hat{y_i} = \arg\max_{j=1...m} P(y_j|x_i). \tag{2.4}$$

To je torej model naivnega Bayesovega klasifikatorja, ki smo ga zgradili na podlage učne množice in napoveduje razred za nove primere, katerih razreda ne poznamo. V nadaljevanju podrobno predstavimo vsak člen izračuna posteriorne verjetnosti.

Razredno pogojna verjetnost

Razredno pogojna verjetnost je verjetnost, da bomo opazili primerek x, če vemo kateremu razredu pripada. V tem delu nastopi predpostavka neodvisnosti. Neodvisnost dveh spremenljivk v resnici pomeni to, da ob poznavanju vrednosti ene spremenljivke ne moremo sklepati o vrednosti druge. Formalno bi lahko rekli:

$$P(x_1x_2...x_n) = \prod_{i=1}^{n} P(x_i). \tag{2.5}$$

Naivni Bayes predpostavi neodvisnost atributov, ne pa neodvisnost atributov od razreda. Zato velja sledeče:

$$P(x|y_j) = P(x_1|y_j)P(x_2|y_j)...P(x_n|y_j) = \prod_{i=1}^{n} P(x_i|y_j). \tag{2.6}$$

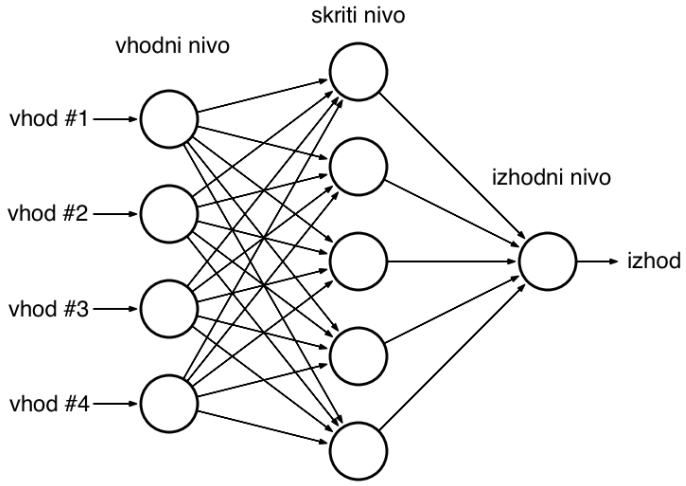
Zgornja enačba je aproksimacija verjetnosti oziroma verjetje, kjer je vsak člen $P(x_i|y_j)$ aproksimiran s pomočjo učne množice kot:

$$\hat{P}(x_i|y_j) = \frac{N_{x_i,y_j}}{N_{y_j}}. \tag{2.7}$$

Člen N_{x_i,y_j} je število primerov v učni množici, kjer ima primer razreda y_j vrednost x_i, N_{y_j} pa je število vseh primerov razreda y_j v učni množici.

Apriorna verjetnost

Apriorna verjetnost je lahko vnaprej podana vrednost s strani domenskega eksperta ali pa jo aproksimiramo iz učne množice. Predstavlja verjetnost,



Slika 2.1: Primer večnivojskega perceptrona, s štirimi vhodnimi nevroni, petimi v skritem nivoju in enim izhodnim.

da naključno izbran primer pripada točno določenemu razredu. Po metodi največjega verjetja velja:

$$\hat{P}(y_j) = \frac{N_{y_j}}{N}. \tag{2.8}$$

Kjer je N_{y_j} število primerov razreda y_j v učni množici in N število vseh primerov v učni množici.

Dokaz

Dokaz je verjetnost, da opazimo primer x neodvisno od razreda. Sicer opazimo, da je dokaz $P(x_i)$ v končnem modelu Naivnega Bayesa v resnici konstanta, ki izhodne vrednosti le skalira, za to ga pogosto ne računamo, saj ne nudi nove informacije. Sicer pa opazimo, da tako zastavljen model deluje le na podatkih, katerih atributi imajo diskretne domene. Celoten problem je mogoče generalizirati tudi na zvezne domene, kot je opisano v [10]. V diplomski nalogi je bila uporabljena implementacija v programeskem jeziku python s knjižnico scikit-learn [19] s privzetimi nastavitvami in parametri.

2.2.2 Nevronske mreže

Nevronske mreže so družina algoritmov, ki so se sposobni učiti tudi bolj kompleksnih, nelinearnih modelov [1]. Temeljijo na osnovni računski enoti imenovani nevron. Nevroni so lahko implementirani na različne načine, cela množica nevronov v pa je povezana tako, da tvori acikličen usmerjen graf, ki mu pravimo mreža. Ker so nevronske mreže precej široka družina algoritmov, se bomo omejili le na osnovno različico imenovano večnivojska usmerjena nevronska mreža ali večnivojski perceptron. V diplomi za večnivojske perceptrone uporabljamo kar izraz nevronska mreža, ker je to edina različica nevronskih mrež, ki jo uporabljamo. Računske operacije v nevronih temeljijo na logistični regresiji, ki jo predstavimo v nadaljevanju.

Logistična regresija

Logistična regresija je algoritem, ki se nauči binarnega klasifikatorja kot linearno funkcijo, ki preslika neko instanco prostora atributov oz. primer x v izhodno vrednost:

$$f(x) = \begin{cases} 1, & \text{če } g(x) \geq 0.5; \\ 0, & \text{sicer.} \end{cases} \tag{2.9}$$

g je navadno logistična funkcija:

$$g(x) = \frac{1}{1 + e^{-w \cdot x}}, \tag{2.10}$$

kjer je w vektor uteži atributov in e Eulerjeva konstanta. Logistična funkcija g preslika primer x v realno vrednost na intervalu $[0, 1]$, kar je analogno verjetnosti, da primer x pripada pozitivnemu razredu. Cilj algoritma je na podlagi učne množice poiskati takšen vektor uteži w, da bo minimizirana kriterijska funkcija:

$$J(w) = \frac{1}{m} \sum_{i=1}^{m} -y_i \log(g(x_i)) - (1 - y_i) \log(1 - g(x_i)), \tag{2.11}$$

kjer je m število primerov v učni množici, y_i razred i-tega primera in x_i i-ti primer. Uteži, ki jih optimiziramo se skrivajo v funkciji g glede na enačbo

(2.9).

Logistična regresija svojo cenovno funkcijo minimizira s spreminjanjem uteži tako, da J s spremembami uteži pada. V eni iteraciji spremeni vsako utež po formuli:

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w), \tag{2.12}$$

kjer je α stopnja učenja, ki jo nastavi ekspert, člen $\frac{\partial}{\partial w_j} J(w)$ pa je odvod cenovne funkcije po j-ti uteži. Na koncu vsake iteracije ponovno izračunamo vrednost J in postopek ustavimo, ko se vrednost bistven no razlikuje od vrednosti v prejšnji iteraciji. Zgornjemu postopku določevanja uteži pravimo gradientni spust.

Večnivojski perceptron

Osnovna enota večnivojskega perceptrona so nevroni, ki so povezani v aciklični usmerjen graf. Vsaka mreža ima dva ali več nivojev, kjer prvemu nivoju pravimo začetni ali vhodni, zadnjemu končni ali izhodni, morebitnim vmesnim pa skriti nivoji.

- **Vhodni nivo** ima natančno n nevronov, kjer je n velikost prostora atributov. i-ti nevron v začetni plasti nosi vrednost i-tega atributa.
- Vsak **skriti nivo** ima poljubno število nevronov. Tudi število nivojev je lahko poljubno. Skriti nivoji omogočajo, da se algoritem nauči kompleksnejših nelinearnih funkcij.
- **Izhodni nivo** ima toliko nevronov kolikor je razredov. Nevronska mreža je za razliko od logistične regresije algoritem, ki brez prilagoditev omogoča uvrščanje v več kot dva razreda.

Nevroni so povezani v mrežo tako, da sosednji nivoji med seboj gradijo polne bipartitne grafe. To pomeni, da je vsak nevron v danem nivoju povezan z vsakim nevronom v sosednjih nivojih, med seboj pa nevroni v istem nivoju niso povezani. Vsaka povezava mreže nosi utež, ki je realno število. Na samem začetku izvajanja algoritma imajo uteži na povezavah naključne vrednosti.

Vrednost nevrona v vsaki naslednji plasti izračunamo tako, da za dani nevron a množimo vrednost nevronov iz prejšnje plasti z njihovimi pripadajočimi utežmi na povezavah z a in vse skupaj seštejemo. Vrednost nevrona a je tako vrednost logistične funkcije z argumentom izračunane vsote:

$$a_i^{(l_k)} = \sum_{j=0}^{|l_{k-1}|} x_j^{(l_{k-1})} w_{ji}^{(l_{k-1})}, \tag{2.13}$$

kjer je $a_i^{(l_k)}$ i-ti nevron v k-tem nivoju l_k, $x_j^{(l_{k-1})}$ j-ti nevron v prejšnem nivoju l_{k-1}, $w_{ji}^{(l_{k-1})}$ povezava med j-tim nevronom v nivoju l_{k-1} in i-tim nevronom v nivoju l_k, $|l_{k-1}|$ pa je velikost oziroma število nevronov v nivoju l_{k-1}. Ko po opisanem postopku izračunamo vrednosti po celotni mreži, dobimo v k-tem nevronu izhodne plasti napoved verjetnosti, da trenuten primer pripada k-temu razredu.
Cilj algoritma je uteži na povezavah nastaviti tako, da bodo napovedi v izhodnem nivoju čimbolj natančne. Tako imajo tudi nevronske mreže svojo kriterijsko funkcijo:

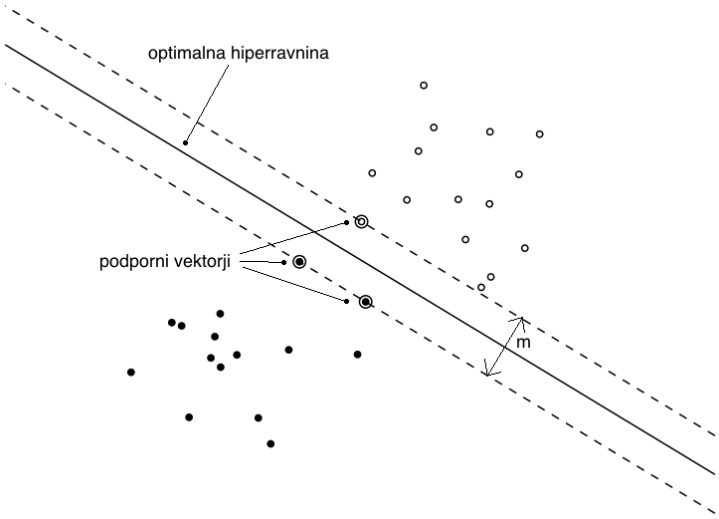
$$J(w) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_i^k \log(h_k(x_i)) + (1 - y_i^k) \log(1 - h_k(x_i)), \tag{2.14}$$

kjer je K število izhodnih nevronov oziroma razredov, m število primerov v učni množici in h vektor iz $\mathbb{R}^K$ in predstavlja izhodni vektor napovedi verjetnosti za vsak razred.

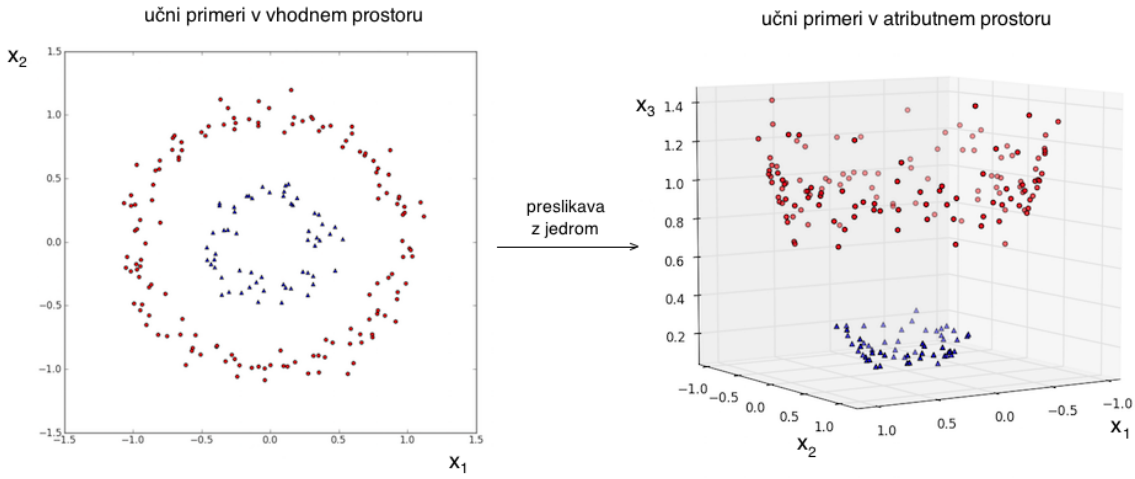
Algoritmu, ki določi uteži povezav v nevronski mreži pravimo vzvratna propagacija, ki pa je precej kompleksnejša kot pri logistični regresiji, saj tu J nima lepih matematičnih lastnosti in zato nima lepega odvoda.

Omenimo, da je logistična regresija le poseben primer takšnega tipa nevronskih mrež, saj je v resnici nevronska mreža brez skritih nivojev in enim nevronom v izhodnem nivoju.

V diplomski nalogi je bila uporabljena implementacija v programskem jeziku python s knjižnico sk-neuralnetwork [20] s privzetimi nastavitvami in parametri.



Slika 2.2: Slika prikazuje optimalno hiperravnino, v tem primeru premico, ki jo najde SVM za prikazan linearno ločljiv problem. Zgoraj so primerki pozitivnega razreda, spodaj negativnega. Na vzporednicah optimalne hiperravnine ležijo dva krat obkroženi podporni vektorji. Vidimo tudi, kako je razdalja med vzporednicama in optimalno hiperravnino največji rob.



Slika 2.3: Slika prikazuje primer kako linearno neločljiv primer z jedrom preslikamo v linearno ločljiv problem v višji dimenziji. Na levi so primeri v vhodnem prostoru, na desni pa primeri v atributnem prostoru. Vidimo, da je problem v atributnem prostoru ločljiv s hiperravnino.

2.2.3 Metoda podpornih vektorjev

Metoda podpornih vektorjev (SVM) je relativno nova metoda strojnega učenja [18], [16]. Večkrat jo primerjamo z nevronskimi mrežami, saj sta se metodi pogosto sposobni naučiti enako kompleksnih funkcij.

Za razliko od nevronski mrež pa ima SVM veliko lepih lastnosti, kar se splošno kaže v hitrem učenju, lažji konfiguraciji in elegantnih teoretičnih lastnostih. Ker je SVM matematično zelo napreden, v diplomskem delu izpustimo globoke detajle njegovih lastnosti.

Za začetek si predstavljamo binarni klasifikacijski problem, kjer je atributni prostor dvodimenzionalen $\mathbb{X} \subseteq \mathbb{R}^2$, razreda pa sta linearno ločljiva. Zaradi razlogov omenjenih spodaj atributnemu prostoru v kontekstu SVM-ja rečemo vhodni prostor. V tako zastavljenem problemu SVM želimo najti premico, ki loči dva razreda. Takšnih premic je pogosto neskončno in klasičen algoritem bi bil zadovoljen že z eno od njih. SVM pravi, da te premice ne dajo enako dobrih rešitev in da obstaja natanko ena, najboljša ločnica. Ta ločnica je

hkrati najbolj oddaljena od obeh razredov zato rečemo, da iščemo premico p, ki maksimizira razdaljo med p in razredoma, ki jih p razmejuje. Intuicija v ozadju je, da lahko najdemo dve p-ju vzporedni premici p_1 in p_2, ki se dotikata vsaj enega primera; prva vzporednica v enem razredu, druga vzporednica pa v drugem. Primeri na teh vzporednicah so tako imenovani podporni vektorji, od koder izhaja tudi ime metode.

Algebraično je ideja maksimizirati dolžino robnega vektorja m, ki je premici p pravokoten vektor v dolžini razdalje med p in njenima vzporednicama $p1$ in $p2$. Naj bo w nek vektor, ki je pravokoten na p. Naša klasifikacijska hipoteza h je:

$$w \cdot x + b \geq 0, \tag{2.15}$$

kjer je x nek primer v učni množici, $w \cdot x$ skalarani produkt med vektorjema w in x, b pa neka konstanta. Dodatna omejitev problema je, da mora za vsak pozitiven primer v učni množici veljati:

$$w \cdot x + b \geq 1, \tag{2.16}$$

za vsakega negativnega pa:

$$w \cdot x + b \leq -1. \tag{2.17}$$

Vrednosti primerov v neodvisnih množicah so seveda lahko tudi na intervalu od -1 to 1, dočim pa za učno množico velja, da takšen primer ne obstaja. Vzporedna interpretacija tega je, da noben učni primer v vhodnem prostoru ne sme biti v pasu med p_1 in p_2. Omenjene omejitve zdaj uporabimo za računanje vektorja w na način, da maksimiziramo rob, oziroma dolžino robnega vektorja $||w||$. Če izračunamo $\frac{w}{||w||}$, dobimo enotski vektor v smeri w, saj je w vzporeden robnemu vektorju m. Ker je skalarni produkt z enotskim vektorjem projekcija na njegovo smer, lahko m najdemo z izračunom skalarnega produkta med razliko dveh podpornih vektorjev v nasprotujočih razredih in $\frac{w}{||w||}$:

$$m = \frac{w}{||w||} \cdot (x_2 - x_1), \tag{2.18}$$

kjer sta x_2 in x_1 podporna vektorja v nasprotnih razredih. Za ta dva vektorja velja:

$$w \cdot x_2 + b = 1, \tag{2.19}$$

$$w \cdot x_1 + b = -1. \tag{2.20}$$

Če enačbo (2.19) odštejemo od enačbe (2.20), dobimo:

$$w \cdot (x_2 - x_1) = 2. \tag{2.21}$$

Tako lahko enačbo (2.21) izrazimo v enačbi (2.18) in dobimo:

$$m = \frac{w}{||w||} \cdot (x_2 - x_1) = \frac{2}{||w||}. \tag{2.22}$$

Če si želimo največjo možno maržo, to pomeni, da moramo minimizirati $||w||$ z upoštevanjem neenakostnih omejitev. Za slednje so najbolj primerni Lagrangejevi multiplikatorji, ki pa sicer operirajo z enakostmi in ne neenakostmi, zato moramo našo metodo prej še prirediti. Po nekaj matematičnih prilagoditvah, bi w izračunali kot vsoto učnih točk:

$$W = \sum_i C_i x_i, \tag{2.23}$$

kjer je C_i neka konstanta, x_i pa so primeri v učni množici. Naslednje vprašanje je kako poiskati C? Osnovno tehniko predstavlja požrešna metoda. Požrešna metoda se dejansko izkaže za optimalno, saj ima naš problem za razliko od nevronskih mrež zelo lepo matematično lastnost imenovano konveksnost. To pomeni, da je le en globalni ekstrem in da nevarnosti ustavljanja v lokalnih ekstremih ni. Tako bo vedno konvergiralo k optimalni rešitvi, za razliko od večnivojskega perceptrona. Izkaže se, da je zato problem tudi računsko obvladljiv. Za izračun C-jev moramo računati skalarne produkte med pari vektorjev v vhodnem prostoru. Večina od teh je ničelnih, tisti, ki jih zares potrebujemo pa so zgolj podporni vektorji.

Zgornja rešitev velja sicer le za linearno ločljive probleme, kar pa lahko rešimo z naslednjo idejo. Vhodni prostor, ki ni linearno ločljiv preslikamo

v nek drug, višjedimenzionalen prostor z uporabo tako imenovane jedrne funckije. Ta ideja je podobna ideji Fourierjeve transformacije. Množico linearno ločimo tako, da:

1. projeciramo primere iz vhodnega prostora v nov prostor imenovan atributni prostor. Atributni prostor ima več dimenzij kot vhodni prostor tako, da lahko primere linearno ločimo.

2. Uporabimo osnovni linearni SVM z atributnim prostorom.

Z jedrom torej problem preslikamo v večdimenzionalnega, kjer so primeri linearno ločljivi, potem pa preslikamo ta prostor nazaj v vhodnega. Več o jedrih najdemo ravno v [18], dočim pa jih v diplomskem delu ne uporabljamo. V odkrivanju znanj iz besedil se pogosto uporablja SVM brez jedra, saj je že vhodni prostor pogosto večdimenzionalen. Čeprav realni problemi niso linearno ločljivi se za linearen SVM uporablja tehnika, ki zna poiskati dobre ločnice ali premice tudi takrat, ko razreda nista linearno ločljiva. V visokodimenzionalnih prostorih ne iščemo premice temveč hiperravnino, za kar ustrezno prilagodimo metodo [18]. V diplomski nalogi je bila uporabljena implementacija algoritma v programskem jeziku python s knjižnico scikit-learn [21] s privzetimi nastavitvami in parametri.

2.3 Merjenje uspešnosti klasifikacije

Intuitivno bi v strojnem učenju uspešnost merili tako, da bi pogledali razmerje pravilnih napovedi nad učno množico. Naivno bi lahko rekli, da je klasifikator, ki se je naučil pravilno uvrstiti čisto vsak primer v učni množici, popoln in bo vedno pravilno napovedoval [11]. Takšno razmišljanje je zavajajoče, saj napovedujemo na istih podatkih na katerih smo se učili. V resnici nimamo nobenega znanja o tem kako se bo klasifikator obnesel na primerih, ki jih še nismo videli v učni množici. Torej je vsak klasifikator pristranski glede na učno množico, saj vemo, da bo dobro napovedoval na njej, ne pa

napovedi / Pravi razredi	razred #1	razred #2	razred #3
razred #1	52	0	8
razred #2	11	79	0
razred #3	0	3	42

Slika 2.4: Slika prikazuje primer kontingenčne matrike za 3 razrede. V stolpcih so pravi razredi, v vrsticah pa napovedani razredi. Prostorček i,j v matriki vsebuje število primerov razreda i, za katere smo napovedali razred j.

nujno na primerih na katerih se ni učil. Temu problemu pravimo preveliko prilagajanje.

Da bi bolje razumeli kako se klasifikator obnese na novih primerih, delimo množico podatkov na učno množico D_{train} in testno množico D_{test}. Na učni množici D_{train} zgradimo klasifikator, s katerim potem napovedujemo na D_{test}. Tako lahko vidimo, kako se klasifikator obnese na podatkih, na katerih se ni učil. Najpogosteje je klasifikator na teh podatkih precej manj točen, kot na podatkih na katerih se je učil. Ko napovemo razrede primerov v D_{Ltest} lahko te napovedi primerjamo s pravilnimi razredi. V nadaljevanju predstavimo nekaj načinov kako to množico napovedi, recimo ji $\hat{y}_{test}$ in množico pravih razredov y_{test} preslikamo v vrednost, ki nam pove kako dobro deluje klasifikator na novih primerih.

napovedi / pravi razredi	pozivitven	negativen
pozivitven	TP	FN
negativen	FP	TN

Slika 2.5: Slika prikazuje primer kontingenčne matrike za 2 razreda. Tu se prostorčki lahko razdelijo v vrednosti TP, FP, FN, TN glede na napovedi in prave razrede primerov.

2.3.1 Kontingenčna matrika

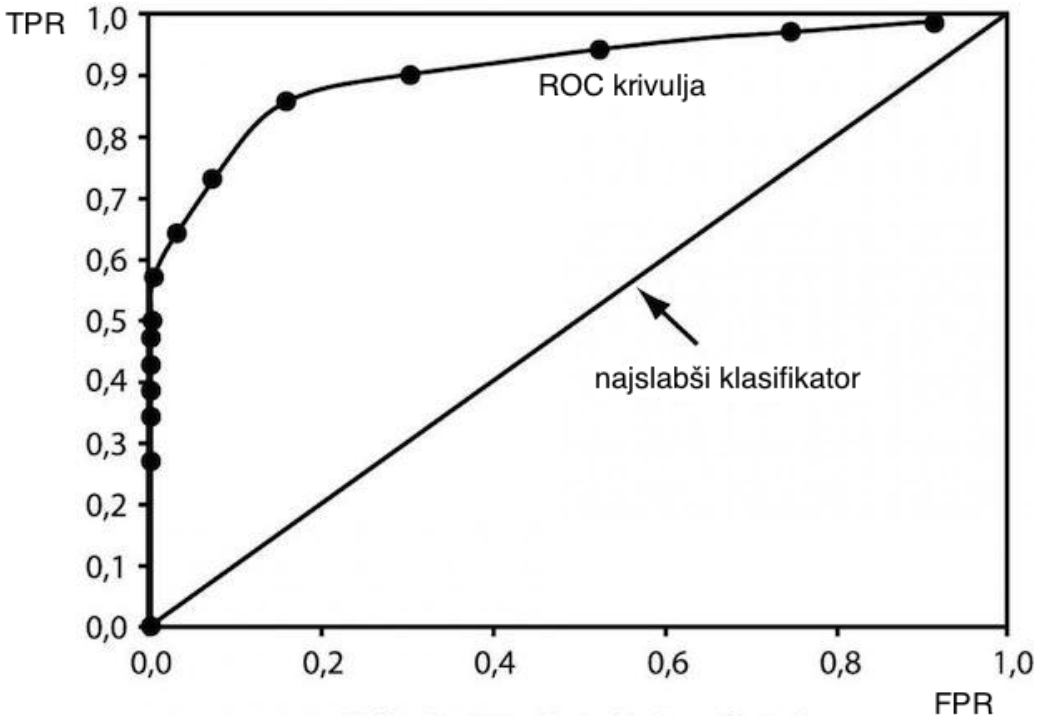
Kontingenčna matrika C je osnova za izpeljavo številnih metrik za merjenje uspešnosti v strojnem učenju. Je matrika, kjer j-ta vrednost v i-ti vrstici matrike predstavlja število primerov, za katere smo napovedali razred j in v resnici pripadajo razredu i. Torej vse vrednosti, kjer $i = j$, predstavljajo pravilne napovedi, vse druge pa napačne.

2.3.2 Klasifikacijska točnost

Klasifikacijsko točnost izračunamo v kontingenčni matriki tako, da sled matrike delimo z vsoto cele matrike:

$$CA = \frac{\sum_{i=0}^{K} C_{ii}}{\sum_{i,j} C_{ij}}. \tag{2.24}$$

Mera nam poda odstotek pravilnih klasifikacij. Ko je porazdelitev razredov razmeroma enakomerna, je klasifikacijska točnost povsem sprejemljiva mera. V primeru, ko je porazdelitev razredov zelo neenakomerna pa lahko dobimo zavajajoče rezultate. Če imamo klasifikacijski problem z dvemi razredi, kjer



Slika 2.6: Na sliki je primer ROC krivulje. Označena je tudi simetrala lihih kvadrantov, ki indicira najslabši možen klasifikator.

je 99% primerov v prvem razredu in le 1% v drugem, potem nam preprost klasifikator, ki vsak primer uvrsti v večinski prvi razred, da klasifikacijsko točnost 0.99. Zdi se, kot da smo problem zelo dobro rešili, dočim nismo odkrili niti enega primera v drugem razredu. Pogosto se namreč izkaže, da je za nas najbolj pomemben razred zelo slabo zastopan v podatkih.

2.3.3 ROC krivulja

ROC krivuljo lahko izračunamo zgolj pri binarnih klasifikacijskih problemih [12]. Če razreda označimo kot pozitivni in negativni razred, potem lahko vsako celico kontingenčne matrike poimenujemo sledeče:

- **TP** - število primerov, ki smo jih napovedali kot pozitivne in so tudi v resnici pozitivni

- **FP** - število primerov, ki smo jih napovedali kot pozitivne in so v resnici negativni
- **FN** - število primerov, ki smo jih napovedali kot negativne, a so v resnici pozitivni
- **TN** - število primerov, ki smo jih napovedali kot negativne in so tudi v resnici negativni

S temi vrednostmi lahko izračunamo dve novi vrednosti:

$$TPR = \frac{TP}{TP + FN}, \tag{2.25}$$

$$FPR = \frac{FP}{TN + FP}. \tag{2.26}$$

TPR je vrednost, ki pove razmerje med številom pozitivnih napovedi, kjer je bil tudi dejanski razred pozitiven in številom vseh pozitivnih napovedi. FPR pa je vrednost, ki pove razmerje med številom pozitivnih napovedi, ki so bile v resnici negativne in številom vseh negativnih napovedi. Intuitivno si želimo čim večji TPR in čim nižji FPR.
Naj bo $t = h(X_{test})$ vektor z izhodi obravnavanega modela h, za vsak primer x v testni množici. Natančneje, i-ti element t_i je izhodna vrednost i-tega primera v testni množici. Pri logistični regresiji so izhodne vrednosti verjetnosti, pri SVM pa oddaljenosti primera od ločitvene hiperravnine. V obeh primerih jih lahko interpretiramo kot hipotetične pragove za končno odločitev, če primer res uvrstimo kot pozitivnega. V tem primeru imamo lahko za vsako vrednost t_i pripadajočo funkcijo:

$$d(x; t_i) = \begin{cases} 1, & \text{če } h(x) \geq t_i, \\ 0, & \text{sicer}, \end{cases} \tag{2.27}$$

kjer je $h(x)$ izhodna vrednost napovedi modela h (za katerega računamo ROC krivuljo) za primer x. Če vsako vrednost v vektorju t preslikamo v nove vrednosti z $d(x; t_i)$ za vsak t_i v t, potem za vsak t_i dobimo različne napovedi iz katerih lahko izračunamo kontingenčno matriko in vrednosti TPR in FPR.

Z zadostnim številom vrednosti za TPR in FPR lahko zgradimo krivuljo $TPR(FPR)$, ki je pravimo ROC krivulja. Površina pod ROC krivuljo, ki ji s kratico pravimo AUC, nam pove kakšna je verjetnost, da bo imel naključno izbrani pozitivni primer večjo izhodno vrednost napovedi kot naključno izbrani negativni primer v testni množici. AUC se obnese precej bolje kot klasifikacijska točnost na problemih z zelo neuravnoteženimi porazdelitvami razredov. Na sliki 2.6 vidimo primer ROC krivulje.

2.4 Metode prečnega preverjanja

V prejšnjem poglavju smo med drugim spoznali tudi problem prekomernega prilagajanja. Večkrat se zgodi, da model naučen na učni množici na njej daje dobre napovedi, na testni množici pa so napovedi slabe. Tak problem lahko odpravimo na primer z uvedbo novih atributov in filtriranjem starih ali pa regularizacijo. Regularizacija je metoda, ki splošči učni množici preveč prilagojeno funkcijo tako, da ta bolje posploši nove primere. Zaradi enostavnosti recimo nastavitvi prostora atributov in parametrom algoritma, kot je na primer tudi regularizacija, kar konfiguracija algoritma. Naivno bi lahko preizkusili več konfiguracij algoritma in na koncu izbrali tisto možnost, ki nam daje najboljše napovedi po neki metriki. Ob taki strategiji se nam hitro zgodi, da konfiguracijo preveč prilagodimo testni množici. Čeprav model daje dobre rezultate na testni množici, bo verjetno ponovno deloval slabše na novih, še ne videnih primerih. V tem podpoglavju predstavimo metodo, ki odpravi ali pa vsaj ublaži omenjen problem.

Prečno preverjanje je splošen način za napovedovanje uspešnosti nekega modela na validacijski množici podatkov [13]. Množico podatkov večkrat različno delimo na učno in testno množico, zgradimo model in izračunamo neko mero uspešnosti nad testno množico. Na koncu vse vrednosti mere povprečimo. Ta povprečena vrednost nam pove kako dobre napovedi lahko lahko pričakujemo za določeno konfiguracijo.

Poznamo več strategij prečnega preverjanja, vse pa se v vsaki delitvi

ohranjajo enako razmerje velikosti učne in testne množice. V grobem se delijo na dve skupini:

- **izčrpno**, kjer preizkusimo vse možne delitve celotne množice na učno in testno;
- **neizčrpno**, kjer preizkusimo le podmnožico vseh možnih delitev celotne množice na učno in testno.

Primer izčrpnega prečnega preverjanja je tako imenovano leave-p-out prečno preverjanje. Tu izberemo p primerov iz celotne množice za testno množico, vsi ostali pa so učna množica. To naredimo na vse možne načine.

Hitro opazimo, da je takšnih delitev zelo veliko. Preizkusiti bi morali $C_p^n = \frac{n!}{p!(n-p)!}$ različnih delitev, kar precej hitro povzroči kombinatorično eksplozijo. Zato se v praksi pogosteje uporablja neizčrpno prečno preverjanje. Predstavimo dve pogosti strategiji prečnega preverjanja:

- K-**kratno prečno preverjanje**, kjer razdelimo celotno množico podatkov na K enakomernih kosov. Iz njih vzamemo enega, ki ga priredimo testni množici, vse ostalo je učna množica. To ponovimo za vse kose. V vsaki od K iteracij zgradimo model in izračunamo vrednost mere točnosti nad testno množico. To vrednost povprečimo ter dobimo pričakovano vrednost za mero točnosti pod trenutno konfiguracijo. Opisana oblika prečnega preverjanja je najbolj pogosta v praksi.
- **Monte Carlo prečno preverjanje**, kjer iz celotne množice vzamemo naključnih K primerov in jih priredimo testni množici, ostalo je učna množica. Zgradimo model in izračunamo novo vrednost za mero točnosti. Slednje ponovimo poljubnokrat, pri čimer se povprečje teh vrednosti vedno bolj približuje pravi povprečni vrednosti.

Prečno preverjanje nam torej ublaži problem prekomernega prilagajanja in pove boljši približek točnosti napovedi, ki jo lahko pričakujemo na neodvisni množici podatkov. V diplomskem delu uporabimo obe strategiji neizčrpnega prečnega preverjanja.

2.4.1 Stratifikacija prečnega preverjanja

Predstavljajmo si neko množico podatkov, v kateri sta dva možna razreda, pozitivni in negativni. Naj bo negativnih 1% primerov, ostali pa naj bodo pozitivni. Če je primerov neskončno in je množica naključno premešana, potem se v primeru delitve množice na več zaporednih podmnožic, porazdelitev razredov v teh podmnožicah ne razlikuje od porazdelitve razredov v celotni množici. Če pa je začetna množica podatkov končna, pa obstaja neničelna verjetnost, da se bodo porazdelitve spremenile. Ta verjetnost narašča s padanjem števila primerov. Omenjeno dejstvo postane nevarno, ko je primerov malo in je porazdelitev razredov zelo neenakomerna.
Predstavljamo si množico 1000 primerov, od katerih je 10 pozitivnih iz katere vzamemo 200 naključnih primerov. Kakšna je verjetnost, da noben od njih ne bo pozitiven?

$$P(\text{ni pozitivnih}) = \prod_{i=0}^{200} \frac{990 - i}{1000 - i} = 0.1062 \tag{2.28}$$

Opazimo, da je verjetnost precej visoka. Pri tako razredno neuravnoteženih problemih zato pogosto v prečno preverjanje vsilimo pogoj, da se mora porazdelitev razreda v podmnožicah prečnega preverjanja ohranjati [22]. To naredimo tako, da množico podatkov najprej razdelimo na pozitivne in negativne, nato pa jih naključno premešamo in razdelimo v enakomerne podmnožice. V vsako množico negativnih primerov vmešamo natanko eno množico pozitivnih primerov. S tem se porazdelitev po razredih v podmnožicah ohranja. Koncept stratifikacije bo zaradi zelo redkih pozitivnih primerov pomemben tudi za problem, ki ga rešujemo.

2.5 Odkrivanje znanj iz besedil

Odkrivanje znanj iz besedil je podpodročje podatkovnega rudarjenja, ki se ukvarja s problemom pridobivanja visokokakovostne informacije iz besedil. Tipični problemi v odkrivanju znanj iz besedil so: gručenje besedil, ekstrak-

cija konceptov in entitet, analiza sentimentov, povzetje dokumentov, kategorizacija besedil in drugi. V diplomski nalogi se ukvarjamo s kategorizacijo besedil, druge vidike odkrivanja znanja iz podatkov pa bralec lahko najde v [14].

2.5.1 Kategorizacija besedil

Kategorizacija besedil [15] se ukvarja z uvrščanjem dokumentov v kategorije. Glede na to, da gre za uvrščanje, problem pogosto rešujemo s klasifikacijskimi algoritmi, ki smo jih v diplomski nalogi že opisali v poglavju 2.2. Problem, ki ostane je kako dokumente opisati kot primere v prostoru atributov in prostoru oznak, kot ju zahtevajo klasifikacijski algoritmi. Osnoven problem je v resnici kako besedila kvantizirati, ga predstaviti kot matematičen vektor oziroma kako ekstrahirati atribute za učenje iz teksta. Za to obstaja več metod, ena najbolj osnovnih in intuitivnih pa je tako imenovana "vreča besed" [16]. Po metodi vreče besed prostor atributov definiramo tako, da poiščemo celotno množico unikatnih besed v učni množici dokumentov. Za nek primer x, i-ti atribut v prostoru atributov vsebuje informacijo o povezavi med i-to unikatno besedo v učni množici dokumentov in tekstom predstavljenim s primerom x. Ta informacija je lahko število pojavitev te besede v tekstu primera x ali pa le logična vrednost, ki pove če se je beseda pojavila ali ne. Popularen tip te informacije je mera TF-IDF [16]. Tipično je TF-IDF utež sestavljena iz dveh delov, TF in IDF. TF meri kako pogosto se beseda pojavi v dokumentu. Ker so dokumenti različnih dolžin, je mogoče da se v dveh dokumentih beseda pojavi enakokrat, vendar je en dokument veliko daljši od drugega. To ustvari iluzijo o tem, da je beseda enako pomembna v obeh dokumentih, čeprav to očitno ni res. Zato število pojavitev pogosto delimo z dolžino dokumenta d.

$$TF(t,d) = \frac{\#\text{ pojavitev besede } t \text{ v dokumentu } d}{|d|} \tag{2.29}$$

IDF ali inverzna dokumentna frekvenca meri kako pomembna je beseda. Ko računamo TF, vsako besedo obravnavamo kot enako pomembno. V slovenskem jeziku opazimo, da se besede kot so "je", "in", "da"pojavljajo zelo

pogosto, vendar nosijo malo informacije. Tako želimo tem besedam znižati uteži z:

$$IDF(t) = \log\Big(\frac{\#\text{ dokumentov}}{\#\text{ dokumentov, ki vsebujejo besedo } t}\Big) \tag{2.30}$$

TF-IDF besede je tako produkt zgornjih termov:

$$TFIDF(t,d) = TF(t,d) \times IDF(t) \tag{2.31}$$

TF-IDF uteži odpravijo omenjene pomankljivosti, ki jih imajo reprezentacije s številom pojavitev besede t v nekem dokumentu. Besedila lahko uvrščamo na sledeč način: Recimo, da imamo podano množico označenih dokumentov D_L in prostor oznak (kategorij) Y.

1. nad D_L zgradimo prostor atributov po principu vreče besed;
2. za vsak dokument d izračunamo TF.IDF vrednosti za vsako besedo v prostoru atributov. S tem zgradimo vektor atributov, ki ga lahko uporabimo za klasifikacijo;
3. izberemo klasifikacijski algoritem in se na pridobljeni množici podatkov naučimo funkcije h, ki bo uvrščala primere v različne kategorije.

2.6 Problem večnega učenja

Neformalna definicija probema večnega učenja je, da je agent sposoben učenja, da se kot ljudje uči več različnih tipov znanja, z leti diverzne in primerno samonadzorovane izkušnje, ter uporablja prej pridobljeno znanje za izboljševanje prihodnega učenja [7]. Problem večnega učenja je formalno definiran sledeče:

$$\mathbb{L} = (L, C) \tag{2.32}$$

$$\text{kjer} \quad L = \{\langle T_i, P_i, E_i\rangle\} \tag{2.33}$$

$$\text{in} \quad C = \{\langle \phi_k, V_k\rangle\} \tag{2.34}$$

Problem večnega učenja $\mathbb{L}$ je sestavljen iz množice učnih nalog L in množice sklopnih omejitev C. Množica učnih nalog L je sestavljena iz trojic $L = \{\langle T_i, P_i, E_i \rangle\}$, kjer je T_i neka naloga, ki se jo želimo naučiti, ali bolj formalno par $\langle X_i, Y_i \rangle$, ki definira domeno in zalogo vrednosti funkcije f_i, ki se jo želimo naučiti.

$$f_i \colon X_i \mapsto Y_i \tag{2.35}$$

P_i je performančna metrika, konkretno preslikava $P_i \colon f_i \mapsto \mathbb{R}$ ki definira optimalno naučeno funkcijo f_i^*:

$$f_i^* = \underset{f_i \in F_i}{\operatorname{argmax}} P_i(f_i), \tag{2.36}$$

kjer je F_i domena hipotez oziroma množica vseh možnih preslikav $X_i \mapsto Y_i$. E_i je tip izkušnje s katero se T_i izboljšuje po P_i. Množica sklopnih omejitev C je sestavljena iz parov $\{\langle \phi_k, V_k \rangle\}$. ϕ_k je funkcija, ki dve ali več učnih nalog L_i preslika v $\mathbb{R}$ in predstavlja stopnjo zadovoljenosti omejitve, V_k pa je vektor indeksov učnih nalog L_i. Agent $\mathbb{A}$, ki rešuje nek problem večnega učenja $\mathbb{L}$ dobi n učnih nalog in proizvede zaporedje rešitev za te učne naloge. S časom se morajo te rešitve izboljševati z ozirom na $P_1, P_2, ..., P_n$ in v stopnji zadoščenosti sklopnih omejitev. Zaradi kompleksnosti NELL-ovega problema večnega učenja, zgornjo teorijo ilustriramo na enostavnejšem primeru.

Recimo, da imamo mobilnega robota s senzorskimi vhodi S in mogočimi akcijami A. Prva naloga $\langle S, A \rangle$, bi lahko bila da se robot nauči izbrati primerno akcijo A iz poljubnega stanja S. Korespondenčna učna naloga $\langle \langle S, A \rangle, P_1, E_1 \rangle$, bi bila naučiti se specifično funkcijo

$$f_1 \colon S \mapsto A \tag{2.37}$$

, ki najhitreje vodi robota v ciljno stanje definirano s P_1. Nauči se z izkušnjo E_1, ki jo dobi s človeško operacijo. Druga učna naloga $\langle \langle S \times A, S \rangle, P_2, E_2 \rangle$ bi pomenila, da robot zna predvideti v katero stanje bo prišel, če opravi neko akcijo a iz stanja s. Z drugimi besedami se učimo funkcijo

$$f_2 : S \times A \mapsto S \tag{2.38}$$

s performančno mero P_2, kjer je E_2 izkušnja robotove avtonomne vožnje po terenu. Za te dve učni nalogi lahko uvedemo sklopno omejitev $\phi(L_1, L_2)$, ki narekuje, da mora f_1 izbrati akcije, ki vodijo optimalno do cilj pa predikcijah f_2. S takim sklopljenjem, damo $\mathbb{A}$ možnost, da se izboljša v učenju ene funkcije s tem, da se izboljša v učenju druge. Učnih nalog ni potrebno reševati simultano. Agent lahko sam dodaja nove učne naloge in nove sklopne omejitve, da se izogne prenehanju izboljševanja. Torej si agent pomaga s proizvodnjo novih učnih nalog, ki mu pomagajo pri celotnem L.

2.7 Aktivno učenje

Spoznali smo, da v nadzorovanem učenju potrebujemo primere, ki so označeni. To pomeni, da imajo poleg podanih vrednosti iz atributnega prostora tudi vrednost ciljne spremenljivke, ki je v primeru klasifikacije pripadajoč razred. Včasih je te razrede lahko pridobiti, pogosto pa izjemno težko. Kot primer povemo, da je lahko označevati elektronko pošto kot neželeno ali želeno, veliko težje pa je označevati sklop bančnih transakcij kot goljufiv ali ne goljufiv. Problem nastane, ker je zaradi visoke cene označevanja podatkov le-teh premalo za izgradnjo modela, ki zna za dan problem dobro generalizirati.
Kot rešitev so bile predlagane nove paradigme za zmanjševanje cene označevanja primerov brez značilne škode za uspešnost učenja. Dve najbolj razširjeni sta:

- **polnadzorovano učenje**, kjer je bistvo, da označene primere izkoristimo za propagiranje njihovih oznak na neoznačene primere. Celo množico označenih in neoznačenih primerov gručimo ter neoznačenim sosedom označenih primerov priredimo iste oznake;

- **aktivno učenje**, kjer želimo v neoznačeni množici najti tiste primere, ki se jih najbolj šplača"označiti [17].

V tem diplomskem delu se posvetimo le aktivnemu učenju.

Recimo neoznačeni množici D_U in označeni množici D_L. Glavni problem aktivnega učenja je kako v D_U izbrati takšne primere, da bo lahko

model naučen na njih maksimiziral svojo napovedno točnost v primerjavi z enostavno rešitvijo, kot je naključno izbiranje primerov iz D_U. Na podlagi modela zgrajenega zgolj z majhno množico D_L bi želeli določiti katere primere v D_U se najbolj splača označiti. Aktivno učenje obravnavamo iz dveh perspektiv:

1. Kako izbrati primere, ki se jih najbolj splača označiti?

2. Kako izbrane primere vrednotimo?

Na drugo vprašanje lahko odgovorimo z dvema kategorijama odgovorov z uporabo enega ali več modelov. Slednje temelji na ideji, da naučimo več modelov na istih podatkih, modeli pa nato glasujejo za razred nekega novega primera. V diplomskem delu gradimo na enem samem modelu. Poizvedba z enim modelom je precej enostavna. Ko nek primer vrednotimo, lahko izkoristimo izhodno verjetnostno porazdelitev nad razredi ali same spremembe v modelu za vrednotenje posameznih primerov. V nadaljevanju si ogledamo kako izbrati primere, ki se jih najbolj splača označiti.

Aktivno učenje temelji na koristnosti u, ki nam pove kako koristen bo nek primer za izboljšavo modela, če ga bomo označili. Za namen definicije takšne mere sta uporabljena dva koncepta: nezanesljivost in korelacija. Prva je kriterij, ki pove kako prepričan je model v svojo napoved, druga pa meri korelacije med samimi neoznačenimi primeri. Glede na slednje delimo aktivno učenje na metode kjer upoštevamo samo nezanesljivost posameznih primerov in jih obravnavamo kot neodvisne ter na metode kjer v obzir vzamemo tudi korelacije med primeri.

2.7.1 Nezanesljivost posameznih primerov

Podano imamo množico neoznačenih primerov D_U, množico označenih primerov D_L in funkcijo nezanesljivosti $f_u(.)$, ki je hkrati mera koristnosti, torej $u(.) = f_u$. Aktivno učenje kjer privzamemo neodvisnost primerov želi zgraditi model z označevanjem po $u(.)$ najbolj informativnih primerov v D_U za

izgradnjo učne množice D_L. S pomočjo D_U in prostora oznak y, f_u definiramo na sledeč način:

$$f_u : D_U \mapsto \mathbb{R} \tag{2.39}$$

Opazimo, da tako definirane metode računajo koristnost primerov v tem, da obravnavajo primere kot neodvisne od ostalih. V grobem se delijo v tri skupine:

- **Nezanesljivostno vzorčenje**. Metode v tej skupini pogosto izkoristijo verjetnostne modele za vrednotenje primera, kot je na primer napoved nekega modela kot verjetnostna porazdelitev nad razredi. Metode se pogosto uporabljajo pri klasifikaciji.

- **Pričakovana dolžina gradienta**. Metode v tej skupini izberejo primere, ki povzročijo maksimalno spremembo v trenutnem modelu. Pogosto se uporablja pri rangirnih funkcijah.

- **Zmanjševaje variance** želi zmanjšati napako modela z izbiranjem primerov, ki imajo najmanjšo varianco. Največji problem tu je visoka časovna kompleksnost, saj je treba za vsak primer zgraditi kvadratno matriko velikosti $K \times K$, kjer je K število parametrov v modelu.

V diplomskem delu bomo obravnavali nezanesljivostno vzorčenje, zato v nadaljevanju predstavimo tri takšne metode.

Metode nezanesljivostnega vzorčenja

Ena od metod nezanesljivostnega vzorčenja je metoda najmanjšega zaupanja s kriterijsko funkcijo:

$$x^*_{LC} = \underset{x}{\operatorname{argmax}}\{1 - P_\Theta(\hat{y}|x)\}. \tag{2.40}$$

Tu je $P_\Theta(\hat{y}|x)$ posteriorna verjetnost najbolj verjetnega razreda $\hat{y}$ po modelu Θ za primer x. Ta metoda preferira primere za katere je trenuten model najmanj prepričan glede razreda. Enačba (2.40) je hkrati tudi najbolj osnovna

metoda nezanesljivostnega vzorčenja. Podobna metoda išče največjo razliko med posteriorno verjetnostjo dveh najbolj verjetnih razredov:

$$x_M^* = \underset{x}{\operatorname{argmin}}\{P_\Theta(\hat{y}_1|x) - P_\Theta(\hat{y}_2|x)\}, \tag{2.41}$$

kjer sta $\hat{y}_1$ in $\hat{y}_2$ prvi in drugi najbolj verjetni razred. Najbolj informativni primeri tu so tisti, ki imajo najmajšo razliko med dvema najbolj verjetnima razreda. Ta metoda se navadno uporablja skupaj z algoritmom SVM. Imenuje se metoda najmanjšega roba. Pristranskost zgornjih dveh metod je to, da ignorirata izhodno porazdelitev za vse ostale razrede razen dveh najbolj verjetnih. Tretja mera, ki v obzir vzame vse elemente te porazdelitve je entropijska mera, ki je definirana kot:

$$x_E^* = \underset{x}{\operatorname{argmax}}\{-\sum_i P_\Theta(\hat{y}_i|x) \log P_\Theta(\hat{y}_i|x)\}. \tag{2.42}$$

Tu je $\hat{y}_i$ posteriorna verjetnost, da primer x pripada razredu i. Vidimo, da ta mera računa entropijo celotne verjetnostne porazdelitve. Ta mera bo v binarnem klasifikacijskem problemu maksimalna takrat, ko bosta oba razreda enako verjetna.

Prvi dve spoznani metodi želita zmanjšati napako klasifikacije, saj izbirata primere, ki pomagajo diskriminirati med specifičnimi razredi. Medtem entropijska mera želi minimizirati logaritmično izgubo modela.

2.7.2 Korelacijsko aktivno učenje

Mnogo študij kaže na to, da aktivno učenje, kjer privzamemo neodvisnost primerov pogosto izbira osamelce. Le-to so primeri, ki so od ostalih znatno oddaljeni. Poleg tega so lahko izbrani primeri tudi redundantni. To je zato, ker v obravnavi nekega primera ne vzamemo v obzir njegove korelacije z drugimi primeri. Za razrešitev tega problema integriramo v našo mero koristnosti tudi funkcijo korelacije.

Če imamo neoznačeno množico D_U in prostor oznak Y, lahko definiramo funkcijo korelacije q_c med dvema primeroma na tri različne načine, glede na

podatke, ki jih upoštevamo pri izračunu:

$$q_c : D_U \times D_U \mapsto \mathbb{R}, \tag{2.43}$$

$$q_c : Y \times Y \mapsto \mathbb{R}, \tag{2.44}$$

$$q_c : (D_U, Y) \times (D_U, Y) \mapsto \mathbb{R}. \tag{2.45}$$

Prvi način, prikazan v enačbi (2.43) izkoristi zgolj množico neoznačenih primerov, drugi, v enačbi (2.44), izkoristi zgolj prostor oznak, tretji (2.45) pa kombinacijo obeh. V diplomskem delu uporabljamo le način predstavljen v enačbi (2.43), zato je v nadeljevanju, ko je omenjen q_c mišljena ta definicija. Za primera z visoko korelacijo pravimo, da sta podobna, za dva z majhno korelacija, pa da sta različna. S tem lahko definiramo tudi korelacijo posameznega primera kot povprečno korelacijo z ostalimi primeri:

$$q_c(x_i) = \frac{1}{|D_U| - 1} \sum_{x_j \in D_U - x_i} q_c(x_i, x_j). \tag{2.46}$$

Enačba predstavlja gostoto primera x_i v D_U. Večja kot je $q_c(x_i)$, večja je gostota okoli x_i. Najbolj reprezentativni primeri v D_U so intuitivno centralne točke z največjo gostoto, ki imajo največjo korelacijo. Ko poznamo funkcijo korelacije q_c jo lahko v mero koristnosti $u(.)$ integriramo kot:

$$u = f_u \times q_c. \tag{2.47}$$

S to terminologijo lahko formaliziramo aktivno učenje z upoštevanjem korelacije med primeri:
Če imamo podano množico neoznačenih primerov D_U in prostor oznak Y, funkcijo koristnosti $u(.) = f_u(.) \times q_c(.)$, kjer je $f_u(.)$ funkcija nezanesljivosti in $q_c(.)$ funkcija korelacije, aktivno učenje z upoštevanjem korelacij v množici D_U poskusi izbrati najbolj informativne primere po funkciji $u(.)$. Takšno obliko aktivnega učenja v grobem delimo na 4 vrste:

- Preiskovanje prek korelacij atributov,
- Preiskovanje prek korelacij oznak,

- Preiskovanje prek korelacij oznak in atributov,
- Preiskovanje prek grafa podatkov.

Preiskovanje prek korelacij oznak ali oznak in atributov se uporablja kadar napovedujemo vektorje oznak. Preiskovanje prek grafa je uporabna kadar smiselno lahko povežemo podatke v graf. Primeri so vozlišča v grafu, povezave pa odražajo odvisnost med krajišči. Na drugi strani se preiskovanje prek korelacij atributov lahko uporablja tudi, kadar napovedujemo le eno oznako. Korelacij izračunamo s pomočjo metod za gručenje, ki temeljijo na meri podobnosti, kar podrobneje predstavimo v nadaljevanju. Opis drugih metod aktivnega učenja najdemo v [17].

Preiskovanje prek korelacij atributov

Smisel aktivnega učenja s korelacijami je, kot smo videli, da funkcijo nekoristnosti f_u utežimo s funkcijo korelacije q_c. Pri preiskovanju prek korelacij atributov je q_c primera povprečna podobnost vsem drugim primerom v neoznačeni množici D_U, kjer podobnost računamo po atributih. Tako najbolj informativne primere izbiramo:

$$x^* = \underset{x}{\operatorname{argmax}}\{f_u(x) \times \Big(\frac{1}{U}\sum_{u=1}^{U} sim(x, x^u)\Big)^{\beta}\} \tag{2.48}$$

Tu je $f_u(x)$ lahko katerakoli od že spoznanih nezanesljivostnih metod, U je velikost množice neoznačenih primerov, β je parameter pomembnosti člena gostote, ki ga nastavi uporabnik. Večji kot bo β, bolj pomemben bo korelacijski člen enačbe. *sim* je funkcija podobnosti, ki definira razdaljo med dvema primeroma. V nadaljevanju si pogledamo najbolj razširjene funkcije podobnosti.

- **Kosinusna podobnost** meri podobnost med dvema primeroma x in x^u kot kosinus kota med pripadajočima vektorjema atributov kot:

$$sim_{cos}(x, x_u) = \frac{\vec{x} \cdot \vec{x_u}}{||\vec{x}||\ ||\vec{x_u}||}. \tag{2.49}$$

Ta mera je primerna za nizkodimenzionalne prostore in je ena najbolj osnovnih mer podobnosti.

- **KL Divergenca** je nesimetrična mera podobnosti, ki zajema razliko med dvema primeroma kot:

$$sim_{KL}(x, x_u) = exp\Big(-\gamma_1 \sum_{j=1}^{N} P(f_j|\vec{(x)}) * \log \frac{P(f_j|\vec{x})}{\gamma_2 P(f_j|\vec{x_u}) + (1-\gamma_2)P(f_j)}\Big), \tag{2.50}$$

kjer γ_1 in γ_2 kontrolirata hitrost divergence. $\vec{x_u}$ je N-dimenzionalen vektor atributov, $P(f_j|\vec{x})$ posteriorna verjetnost vsebovanosti atributa f_j v celi množici podatkov, $P(f_j)$ pa apriorna verjetnost opažanja atributa f_j. KL Divergenca se pogosto uporablja pri informacijskem poizvedovanju in prepoznavanju imenskih entitet.

- **Gaussova podobnost** je eksponentna mera, ki ocenjuje gostoto informacije in je neke vrste eksponentna Evklidska razdalja, ki upošteva vse razdalje po vsakemu atributu.

$$sim_{gauss}(x, x_u) = exp\Big(-\sum_{j=1}^{J} \frac{(\vec{x^j} - \vec{x_u^j})^2}{\alpha^2}\Big). \tag{2.51}$$

α^2 je tu varianca normalne porazdelitve.

V formuli za koristnost u lahko uporabimo katerokoli mero podobnosti in upoštevamo korelacijo med primeri. Veliko študij kaže tudi na to, da korelacije pripomorejo k temu, da res izbiramo bolj reprezentativne primere iz množice neoznačenih primerov v primerjavi z upoštevanjem zgolj nezanesljivosti. Povedati pa moramo tudi, da je iskanje korelacij pri večjih množicah neoznačenih primerov zelo časovno zahtevno.

2.8 Uporabljene tehnologije

Cilj diplomskega dela je povezava dveh sistemov, kjer eden pomaga drugemu pridobiti nove, koristne metapodatke. V naslednjih dveh poglavjih predsta-

vimo omenjena sistema, kasneje pa predlagamo konkretno rešitev za njuno povezavo.

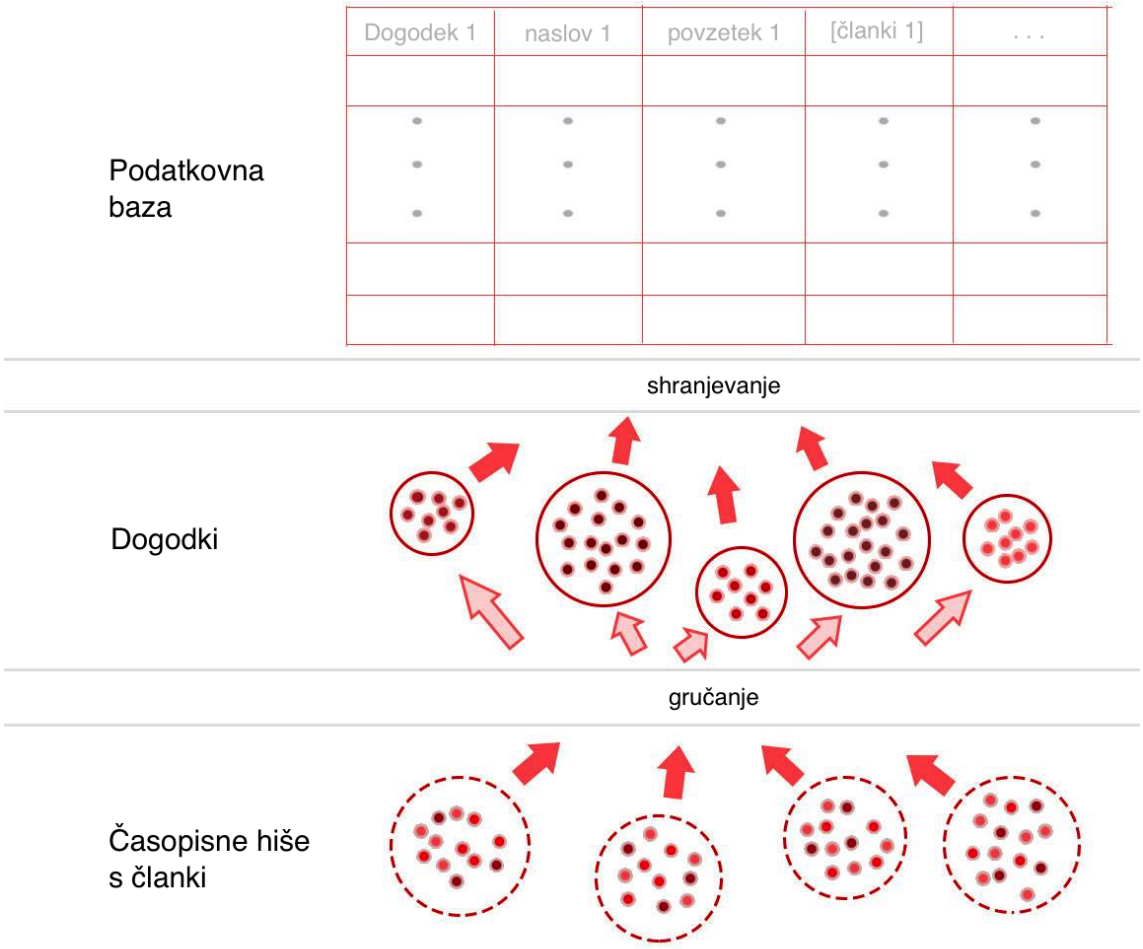
2.8.1 Event Registry

Na tisoče člankov se objavi v časopisnih hišah širom sveta vsak dan. Napisani so v različnih jezikih in diskutirajo vse mogoče teme. Velik procent jih opisuje pretekle, sedanje ali prihodnje svetovne dogodke. Čeprav ni splošno sprejete definicije dogodka, je intuitivna definicija, da je dogodek karkoli značilnega, kar se je zgodilo na svetu. Primer dogodka je tudi rušenje WTC v New Yorku, 11. septembra 2001. Čeprav ljudi zanimajo dogodki, pa je dejanska enota vsebine navadno časopisni članek. Čeprav članki nosijo informacijo o čem gre, kdo je vključen, kdaj se je nekaj zgodilo in podobno, so te informacije skrite v besedilu in se od človeka pričakuje, da jih ekstrahira sam z branjem članka.

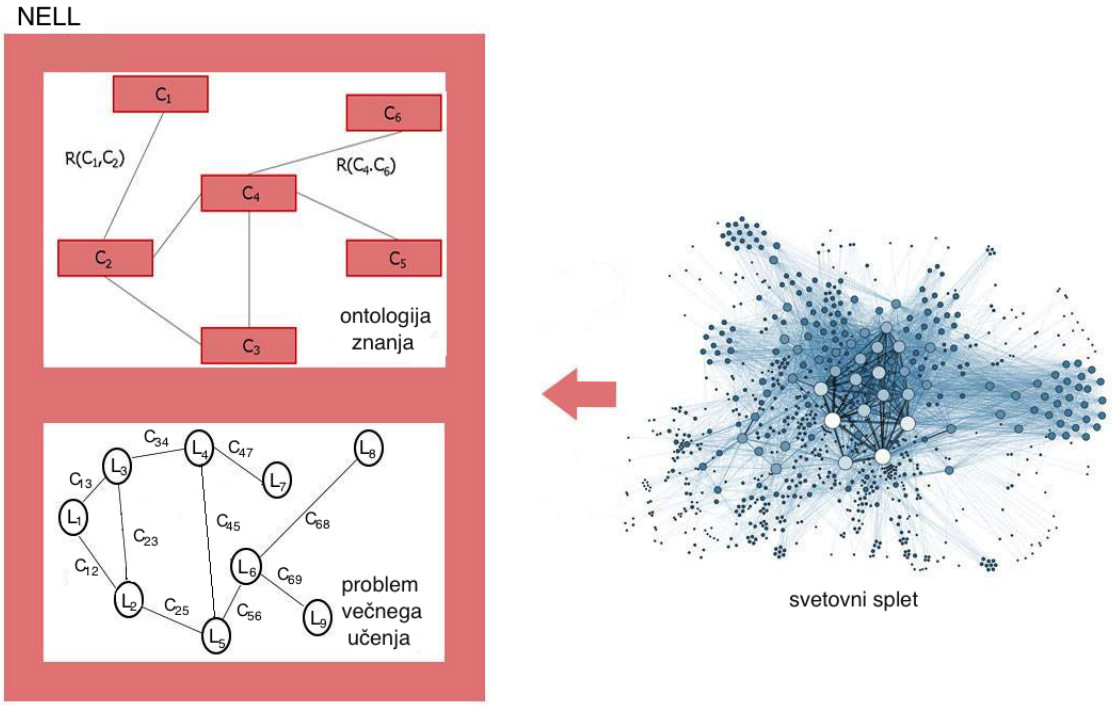
Zgornje je motivacija za nastanek sistema EventRegistry [8]. Sistem nabira članke iz tisoče različnih časopisnih hiš in v njih prepoznava dogodke. Članke, ki so v različnih jezikih in opisujejo isti dogodek, sistem gruči in jih predstavi kot en dogodek. Iz njih ekstrahira informacije o dogodku iz različnih vidikov, kar je koristno saj so posamezni članki lahko pristranski. S tem lahko dogodek vidimo bolj nepristransko kot, če upoštevamo le en vir. Najdene dogodke sistem hrani v podatkovni bazi in omogoči uporabnikom dostop do njih. Ti do dogodkov dostopajo s poizvedbami po entitetah, lokaciji, datumu ali temi. Za vsak dogodek je moč dobiti tudi izvorne članke ali sorodne dogodke. Sistem EventRegistry od leta 2013 spremlja dogodke časopisnih hiš po celem svetu.

2.8.2 Never Ending Language Learner

Never Ending Language Learner (NELL) [7] je učeči se agent, katerega naloga je naučiti se brati splet. NELL ima podano ontologijo, ki definira kategorije (npr. Država, Predsednik) in binarne relacije (npr. JePredse-



Slika 2.7: Slika prikazuje osnovno idejo sistema EventRegistry. V spodnjem nivoju gruče majhnih krogcev predstavljajo časopisne hiše, sami krogci pa so članki, ki so različnih barv, kar predstavlja, da so v člankih različni dogodki. V srednjem nivoju se članki gručijo v dogodke, dočim so kroglice enake brave, saj opisujejo enake dogodke. V zgodnjem nivoju pa se dogodki in izračuni dogodkov, entitete v dogodkih in ostalo shranjuje v podatkovno bazo.



Slika 2.8: Slika prikazuje osnovno idejo sistema NELL. Sistem bere podatke iz spleta, kar se nauči preko problema večnega učenja, ki je predstavljen s spodnjim delom škatle na levi in shranjuje naučeno znanje v ontologiji znanja, predstavljeni v zgornjem delu škatle. Problem večnega učenja je predstavljen z vozlišči, ki so učne naloge L_i in povezavami C med njimi, kjer C_{ij} predstavlja sklopno omejitev med učnima nalogama L_1 in L_2. Ontologija znanja je predstavljena s koncepti C_i, ki jih povezujejo relacije R, kjer je povezava med $R(C_i, C_j)$ relacija, ki sprejema instance konceptov C_i in C_j.

dnikDržave(x,y)). Podano ima tudi majhno množico primerov za vsako kategorijo (npr. Slovenija in Hrvaška za kategorijo Država) s katero dobi začetni občutek za smer učenja. Z branjem spleta nato želi razširiti to ontologijo z ekstrahiranjem novih verjetij (npr. JePredsednikDržave(Obama, Amerika)) na podlagi starih verjetij in novega znanja, ter izbrisati stara verjetja, ki so se izkazala za napačna. Vsak dan se nauči brati bolje kot prejšnji dan.

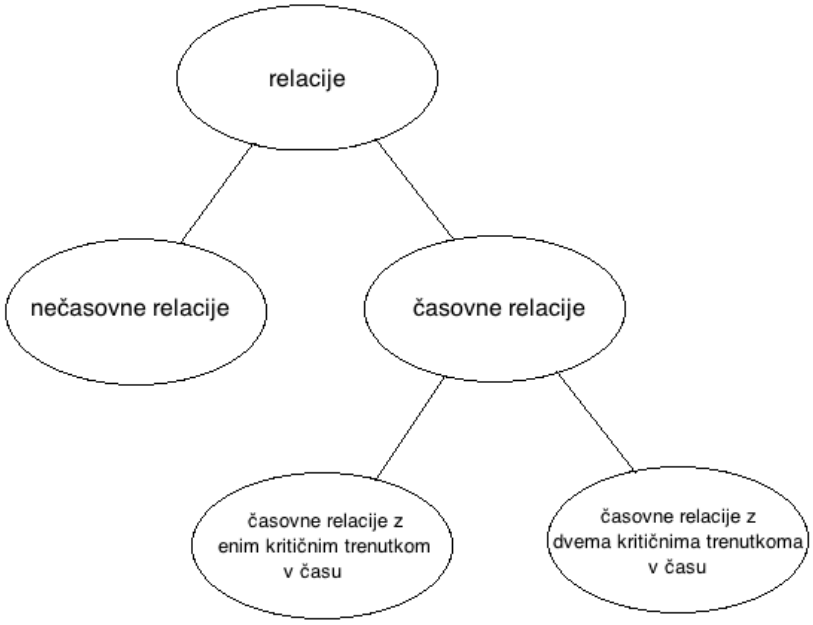
Svojo nalogo NELL opravlja z reševanjem konkretne instance problema večnega učenja. Njegove učne naloge zajemajo kasifikacijo v kategorije, klasifikacijo relacij, resolucijo entitet in nekatere druge. S sklopnimi omejitvami, kot je na primer sklapljanje podmnožic z nadmnožicami, poskrbi za pogoj večnega učenja, da z učenjem ene učne naloge lahko pomagamo učenju druge učne naloge. Naučena verjetja NELL izpostavi v bazi znanja, kjer so shranjena verjetja glede konceptov in binarnih relacij. Za koncepte in relacije imamo navedene instance, ki jih je NELL izluščil z branjem spleta, za vsako instanco pa so podani tudi metapodatki o času, ko je NELL dano verjetje izluščil, o stopnji verjetja in podobno. NELL bere splet že od januarja leta 2010 in je spoznal že več kot 80 milijonov verjetij.

Poglavje 3

Ekstrakcija časovnega znanja

3.1 Problem časovnih relacij

V prejšnem poglavju smo spoznali metode in tehnologije, ki jih uporabljamo tokom dela v diplomski nalogi. Spoznali smo, da je EventRegistry sistem, ki nam nudi opise dogodkov, ki so se zgodili v svetu, NELL pa je sistem, ki skozi branje spleta spoznava svet in se o njem uči. Eden temeljnih problemov sistema NELL je, da nima občutka za čas. V tem kontekstu to pomeni, da se uči nekih dejstev o svetu v obliki spoznavanja konceptov in relacij med njimi, v resnici pa ne ve kdaj so ta dejstva dejansko veljala ali sploh še veljajo. Primer manifestacije tega problema je recimo, da sistem verjame, da mesto predsednika države trenutno pripada vsem predsednikom, ki jih je ta država imela v zgodovini svojega obstoja. Še en tak primer je, da misli, da ima nek človek, ki se je tokom življenja veliko poročal in ločeval z različnimi parterji, v tem trenutku veliko zakonskih partnerjev. Čeprav vemo, da obstajajo tudi taki primeri, jih je dejansko veliko manj kot to misli NELL. Podobnih primerov pomanjkanja občutka za čas je še veliko. Ne smemo pa pozabiti, da je veliko relacij brezčasnih. Primer takšne relacije je recimo JeŠtevilo(42). Ta relacija je veljala od vekomaj in bo veljala vedno v prihodnosti. Za lažjo obravnavo problema v nadaljevanju podamo delovne definicije konceptov, ki jih uporabljamo tokom diplomske naloge.



Slika 3.1: Slika prikazuje kako se relacije delijo glede na odvisnost od časa. Relacije se najprej delijo na časovne in nečasovne, časovne pa se naprej delijo na relacije z enim kritičnim trenutkom v času in takšne z dvema.

- **Koncept** je abstrakcija ali generalizacija izkušenj ali rezultat transformacije obstoječih idej. (npr. Sadež, Žival, Mesto, ...).

- **Instanca koncepta** je konkreten primerek koncepta. Za koncept Sadež je to na primer jabolko.

- **Relacija** je binarna preslikava $r(c_1, c_2, ..., c_n)$, ki zaporedje konceptov c_i preslika v vrednost 1 (resnično) ali 0 (neresnično). Primer relacije je jeŠtevilo(x).

- **Instanca relacije** je konkreten primerek relacije, kjer so instance konceptov, ki se pojavijo kot argumenti relacije ter vrednost relacije znani. Primer instance relacije je jeŠtevilo(42).

- **Časovna relacija** $r(c_1, ..., c_n; t)$ je relacija, ki lahko isto zaporedje konceptov c_i preslika v različne vrednosti ob različnih trenutkih v času.

- **Dogodek** je karkoli, kar se je zgodilo v svetu in ima nek signifikanten pomen.

Omembe vredno je, da so zgoraj podane delovne definicije. Težko je reči kaj je v resnici časovna relacija in v takšnih okoliščinah lahko hitro zaidemo v ontološke vode. Za namene tega diplomskega dela so takšne definicije zadostne.

Realen raziskovalni problem predstavlja kako med vsemi relacijami vseeno ločiti med časovnimi in nečasovnimi relacijami. Na podlagi meta-podatkov, ki jih ponuja NELL za vsako relacijo slednje zaenkrat ni mogoče. Problem rešujejo na univerzi Carnegie Mellon, vendar še niso našli klasifikatorja, ki bi obe skupini relacij znal zanesljivo ločiti. Zato v diplomski nalogi predpostavimo, da poznamo relacije, ki so časovne. Spremenljivost teh relacij bi le stežka iskali preko funkcije, ki je odvisna od časa in zna za vsako točko v času napovedati točno vrednost relacije za določeno zaporedje konceptov. Te spremembe pa lahko iščemo z ekstrahiranjem informacij, skritih v besedilih. Problem nastane, ker ne vemo katera besedila brati, za pridobitev potrebnih

informacij za odkritje spremembe. Sistem EventRegistry omogoča poizvedbe po dogodkih, kjer lahko iščemo besedila, navezujoča se na točno določene koncepte. Tako bi lahko za vsako relacijo opravili poizvedbo v EventRegistry, če bi vedeli kateri koncepti so indikativni za spremembo veljave neke relacije. Spremembo instance relacije povzroči začetek, oziroma sprememba vrednosti instance relacije iz 0 v 1, ali pa konec, kjer se vrednost spremeni iz 1 v 0. Obstajajo seveda tudi časovne relacije, ki se začnejo in končajo ob istem trenutku v času. Primer take je AtletZmagaTekmovanje(Atlet,Tekmovanje). Atlet zmaga tekmovanje in relacija velja le v tem trenutku.

Torej v nekaterih primerih iščemo le dejstvo, da se je nekaj zgodilo in vemo, da se je hitro za tem tudi končalo, v nekaterih primerih pa sta začetek in konec povsem različna in časovno razmejena dogodka. Ne le, da se relacije delijo v časovne in nečasovne, temveč se časovne delijo tudi na takšne z enim kritičnim trenutkom v času in dvema trenutkoma. Tako celoten problem lahko zapišemo kot: **"Kako za instance vsake časovne relacije najti ključne časovne točke, ob katerih se spremenijo njihove vrednosti?"**

3.2 Povezava med sistemom

Če iščemo ključne trenutke v času za neko relacijo s poizvedovanjem po dogodkih, bi nam olajšalo delo, če bi vedeli kako v jeziku sistema EvntRegistry povedati vsaj okvirno katere dogodke si želimo analizirati. Kot že omenjeno, EventRegistry poizvedbe sprejmejo množico konceptov in potem vrnejo dogodke, ki se navezujejo na te koncepte. Če sistemu EventRegisry ne bi podali referenčnih konceptov, bi morali pregledati zelo veliko dogodkov, da bi našli tiste, ki jih želimo.

Zato predpostavimo, da relacije in njihove pripadajoče tipe ključnih točk poznamo množice kjučnih konceptov, s katerimi poizvedujemo v EventRegistry in nam zožajo področje iskanja v podatkovni bazi sistema EventRegistry. Vsak tip ključne točke ima različno referenčno množico. Referenčne množice časovnih relacij z dvema trenutkoma v času so:

- **začetna množica** S - množica, ki jo imajo časovne relacije r z dvema ključnima časovnima točkama. Dogodki, ki so rezultat poizvedbe s S_r lahko vsebujejo informacijo o začetku veljave neke instance relacije r,
- **končna množica** F - množica, ki jo imajo časovne relacije r z dvema ključnima časovnima točkama. Dogodki, ki so rezultat poizvedbe z F_r lahko vsebujejo informacijo o koncu veljave neke instance relacije r.

Referenčne množice časovnih relacij z enim trenutkom v času:

- **dogodkovna množica** H (happening) - edina referenčna množica časovnih relacij z eno ključno točko. Dogodki, ki so rezultat poizvedbe s H lahko vsebujejo informacijo o tem, da je relacija veljaja v eni točki v času.

Če iščemo ključne trenutke za neko relacijo r s poizvedovanjem po dogodkih, potem:

1. V EventRegistry opravimo poizvedbo z množico S_r ali F_r in pridobimo množico dogodkov D.
2. V D poiščemo tiste dogodke, ki vsebujejo informacije o ključnih trenutki v času relacije.
3. Določimo kako in katere instance relacije dogodki spremenijo.
4. Rezultat dodamo pripadajočim instancam relacije r v NELL bazi.

V diplomski nalogi se primarno ukvarjamo s prvima dvema točkama tega postopka, dočim pa predstavimo tudi delno rešitev za tretjega. Čeprav se cel postopek zdi zelo enostaven, se izkaže, da so ta opravila zelo zahtevna za avtomatizacijo, če jih želimo opraviti z visoko zanesljivostjo. Prav tako se omejimo na časovne relacije z dvema kritičnima trenutkoma, konkretno na relacijo $ImaZakonca(x, y)$, pri tem pa rešitev zasnujemo dovolj splošno, da jo je moč posplošiti na druge časovne relacije z dvema kritičnima trenutkoma v času.3

3.3 Omejen klasifikacijski problem

V diplomski nalogi se omejimo na časovno relacijo ImaZakonca(x,y). Med dogodki v EventRegistry želimo najti tiste, ki vsebujejo informacije o ključnih časovnih točkah relacije. Predpostavimo tudi, da imamo podani referenčni množici $S = \{$"Poroka"$\}$ in $F = \{$"Ločitev"$\}$. Iskanje zastavimo kot binarni klasifikacijski problem, kjer so množica podatkov opisi dogodkov, ki vsebujejo naslov in povzetek dogodkain jih želimo uvrstiti v dva razreda: pozitivni, ki so tisti dogodki, ki vsebujejo informacijo o kritičnih trenutkih v času in negativni, tisti ki ne vsebujejo te informacije.

Izkaže se, da je pozitivnih dogodkov izredno malo. To je povsem logično, saj se ljudje ne poročajo in ločujejo vsak dan. Vsaj za večino se to zgodi le enkrat v življenju. Zaradi neuravnoteženosti porazdelitve razredov v začetnem problemu množico podatkov zelo omejimo. V tej stopnji je cilj zgolj, da vemo, kako težak je problem. Če bi se problem tudi s tako veliko pristranskostjo in specifičnostjo izkazal za zelo težak, bi pomenilo, da je problem trenutno pretežak in ta pot ne obeta dobrih rezultatov. Tako smo množico podatkov skrčili na naslednji način:

- poleg poizvedbe po množicah S in F smo poizvedovali tudi po konceptu osebe, sicer po konkretnih instancah osebe, za katere smo vedeli, da so se v zadnjem času ločili ali poročili; množico teh oseb označimo s P, Funkcija $erq(.)$ pa vrne rezultat poizvedbe v EventRegistry po neki množici konceptov;

- opravili smo poizvedbe po unijah $S \cup \{p\}$, ter $F \cup \{p\}$ za vsak $p \in P$. Tako smo dobili množico primerov D:

$$D = \bigcup_{p \in P} (erq(S \cup \{p\}) \cup erq(F \cup \{p\})); \tag{3.1}$$

- da bi množico D še bolj skrčili, smo množico filtrirali z regularnim izrazom, ki zahteva, da mora povzetek dogodka vsebovati vsaj eno od predpon v množici {“marr”, “divorc”, “wed”, “husband”, “spouse”}, ki

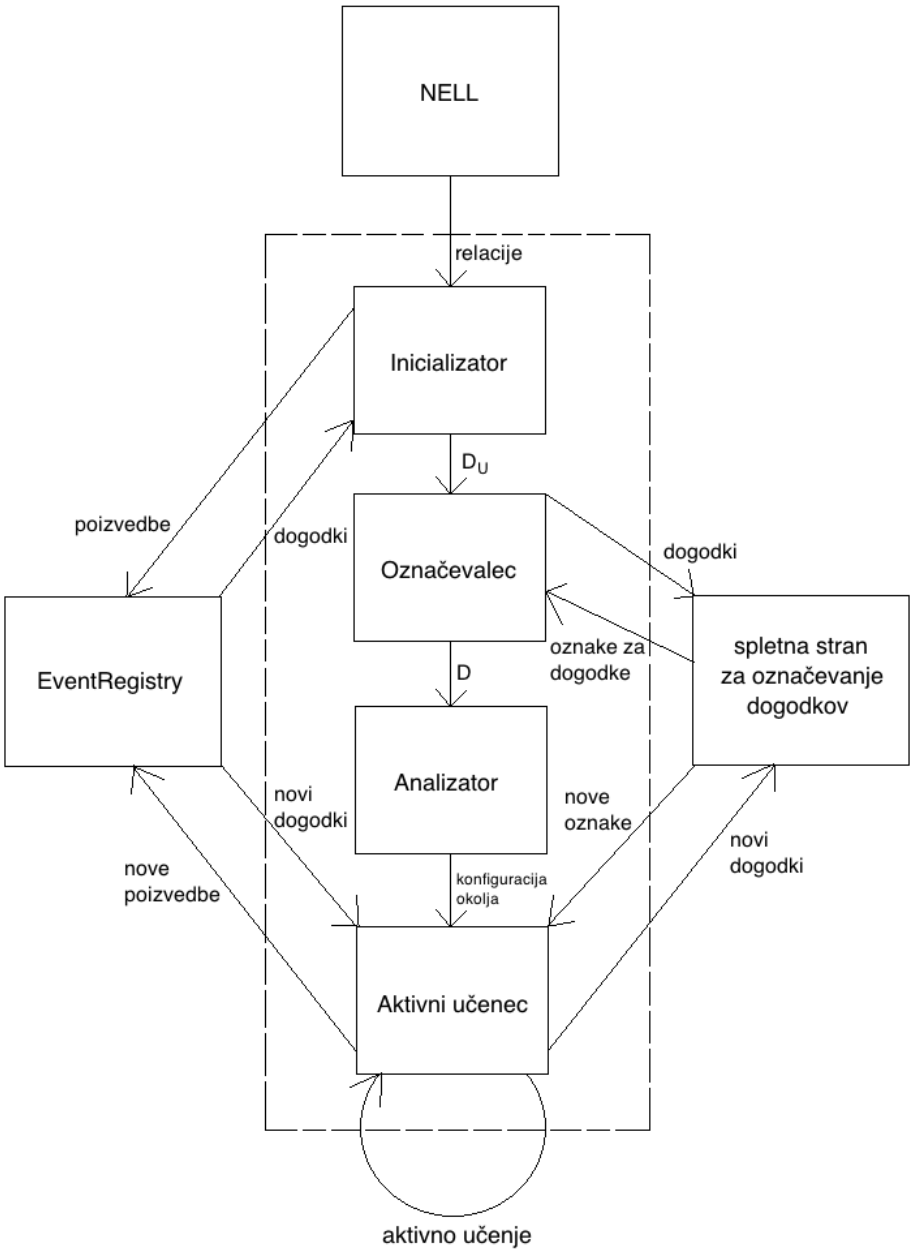
so predpone angleških besed “marriage”, “divorce”, “wedding”, “husband”, “spouse” ter njihovih izpeljank, ki so indikativne za dogodek poroke ali ločitve.

Tako smo množico zmanjšali na 225 primerov. Ker klasifikacijski algoritmi zahtevajo, da so primeri označeni, smo nato primere sami označili. Izkazalo se je, da je bila množica še zmeraj zelo neuravnotežena: 91% primerov je bilo negativnih in le 9% pozitivnih. Problem smo reševali s klasifikacijskim algoritmom SVM, predstavljenem v poglavju 2.2.3, z linearnim jedrom in privzetimi parametri. Uspešnost smo merili z mero AUC, predstavljeno v poglavju 2.3.3 in Monte Carlo prečnim preverjanjem, ki je predstavljeno v poglavju 2.4. V 100 iteracijah Monte Carlo prečnega preverjanja smo dobili povprečen AUC in standardni odklon: $\overline{AUC} = 0.9045$, $\sigma = 0.072$. Rezultati izgledajo precej obetavni. Vrednost AUC pomeni, da če izberemo naključen primer iz množice pozitivnih in naključen primer iz množice negativnih v neki nepoznani množici, potem bo klasifikator v 90% primerih dodelil pozitivnemu višjo oceno kot negativnemu.

3.4 Predlog splošne rešitve

Po obetavnih rezultatih si želimo vsaj okvirno obliko splošne rešitve za prepoznavanje dogodkov z informacijo o ključnih trenutkih v času. Za začetek prepoznamo velik problem. Za klasifikacijo potrebujemo označene primere, ki pa jih je zelo drago pridobiti. Primere mora označevati človek, ekspert, ki je počasen in drag. Po naših izkušnjah se za ta problem primeri označujejo približno s hitrostjo 200 primerov na uro, kar je izjemno počasi.

Problem dragega označevanja nam vsaj nekoliko ublaži aktivno učenje. Čeprav se primeri ne označujejo nič hitreje, nam aktivno učenje z izbiranjem najbolj informativnih primerov za označevanje potencialno zmanjša količino primerov potrebnih za izgradnjo dovolj zanesljivega klasifikatorja. Zaradi tega splošno rešitev oblikujemo pretežno okoli ideje aktivnega učenja. V nadaljevanju predstavimo predlagano rešitev, ki se izvaja neodvisno za vsako



Slika 3.2: Slika je ilustracija predlaganega splošnega sistema. Na vrhu NELL pošlje relacije inicializatorju, ta po njih poizvede v EventRegistry, ki mu pošlje nazaj množico dogodkov. Inicializator preda dogodke označevalcu kot množico D_U neoznačenih primerov oziroma dogodkov. Označevalec pošlje dogodke v D_U na spletno stran za označevanje dogodkov, kjer eksperti označujejo dogodke in jih pošiljajo nazaj označevalcu. Ko so dogodki označeni označevalec pošlje označeno množico D Analizatorju, ki aktivnemu učencu nato preda najboljšo konfiguracijo okolja. Aktivni učenec se nato izvaja v nedogled in poizveduje po novih dogodkih v EventRegistry, jih pošilja nas spletno stran, od nje sprejema oznake, ter se neprestano aktivno uči.

časovno relacijo. Sistem zato predstavimo za poljubno časovno relacijo r. Splošni sistem sestoji iz štirih podsistemov:

Inicializator

Inicializator je zadolžen za pridobitev začetne množice neoznačenih dogodkov za relacijo r. To doseže tako, da koncepte, ki se pojavijo kot argumenti v instancah relacije r sistema NELL, spari ločeno z množico S_r in F_r posebej in opravi poizvedbo nad sistemom EventRegistry. Iz te velike množice nato naključno izbere dovolj veliko število dogodkov, da lahko po označevanju zgradimo začetni klasifikator. Izhod tega podsistema je začetna neoznačena množica D_U.

Označevalec

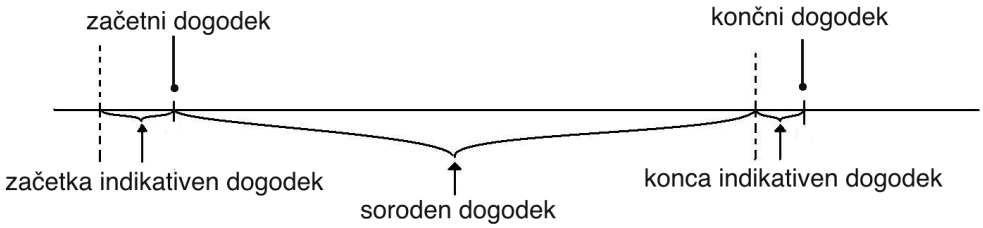
Označevalec poskrbi za to, da se primeri v D_U označijo. Komunicira s spletno stranjo kamor pošilja neoznačene primere. Človeški eksperti na spletni strani označujejo primere, oznake pa se pošiljajo nazaj označevalcu. Ko imamo oznak dovolj, označevalec preda označeno množico D naslednjemu delu sistema.

Analizator

V tem delu izkoristimo množico D za iskanje dobre konfiguracije za aktivno učenje. S konfiguracijo mislimo skupek strategije aktivnega učenja, klasifikacijskega algoritma in njegovih parametrov, ter prostora atributov. Izhod analizatorja je konfiguracija okolja.

Aktivni učenec

Aktivni učenec se izvaja v nedogled. Z naučeno konfiguracijo zgradimo začetni model, s katerim po izbrani strategiji aktivnega učenja izbiramo nove primere v neoznačeni množici. Te primere zopet pošljemo na spletno stran, kjer jih eksperti označijo. Na določen časovni interval, ponovno zgradimo



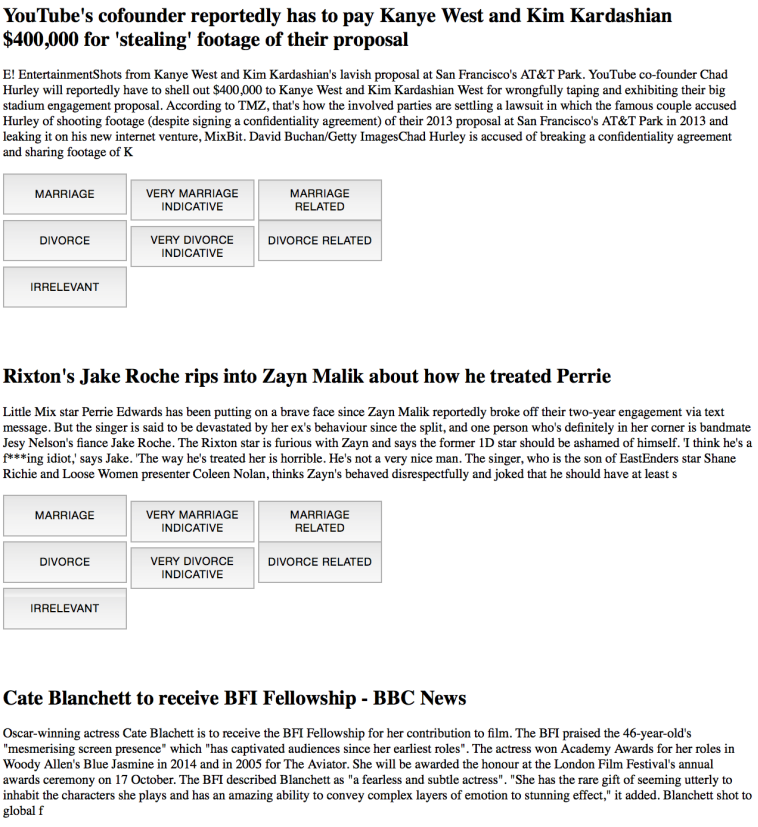
Slika 3.3: Slika prikazuje umestitev relevantnih dogodkov na časovnico, za poljubno relacijo z dvema referenčnima točkama.

model z novimi označenimi primeri. To ponavljamo dokler model ni dovolj zanesljiv.

3.4.1 Označevanje primerov

V časovnih relacijah z dvema kritičnima trenutkoma v času lahko relevantne dogodke delimo na več skupin kot le na dogodke, ki vsebujejo informacijo o časovni točki in dogodke, ki je ne vsebujejo. V diplomski nalogi smo poskusili najti najbolj splošno delitev za relacijo ImaZakonca(x,y), z mislijo o generalizaciji tudi na druge relacije. Dobljene dogodke smo razdelili v 4 skupine, ki jih tudi prikažemo umeščene na časovnici na sliki 3.3. Dogodek, ki spada v skupino:

- **Poroka ali ločitev** je dogodek, ki mora nositi informacijo o konkretni poroki oziroma ločitvi med dvema osebama. Če ni čisto razločljivo iz



Slika 3.4: Slika prikazuje spletno stran, kjer eksperti označujejo podatke. Strežnik na spletno stran pošlje primere, eksperti pa prek spletne strani označujejo primere, katerih oznake se pošiljajo nazaj na strežnik.

dogodka, da sta se v njem dve osebi poročili oziroma ločili, potem ta dogodek ni tega tipa.

- **Indikacija poroke ali ločitve** je dogodek, ki mora nakazovati na to, da se bo poroka ali ločitev verjetno zgodila v bližnji prihodnosti.
- **Sorodno poroki ali ločitvi** je dogodek za katerega je razvidno, da se navezuje na poroko ali ločitev, stanje ločenosti, poročenosti ali katerokoli zadevo povezano s tema dvema konceptoma, vendar ni v zgodnjih dveh skupinah.
- **Nerelevantno** je dogodek, ki ne spada v nobeno od zgornjih treh skupin.

Takšno delitev lahko tudi generaliziramo v:

- **Dogodek z informacijo o kritičnem trenutku v času**
- **Indikativen dogodek** je dogodek, ki nakazuje na to, da je kritični trenutek v času v bližnji prihodnosti.
- **Soroden dogodek** je dogodek, ki se prav tako navezuje na isto temo kot zgornja dva tipa, vendar pa ne sodi tja.
- **Nerelevantno**

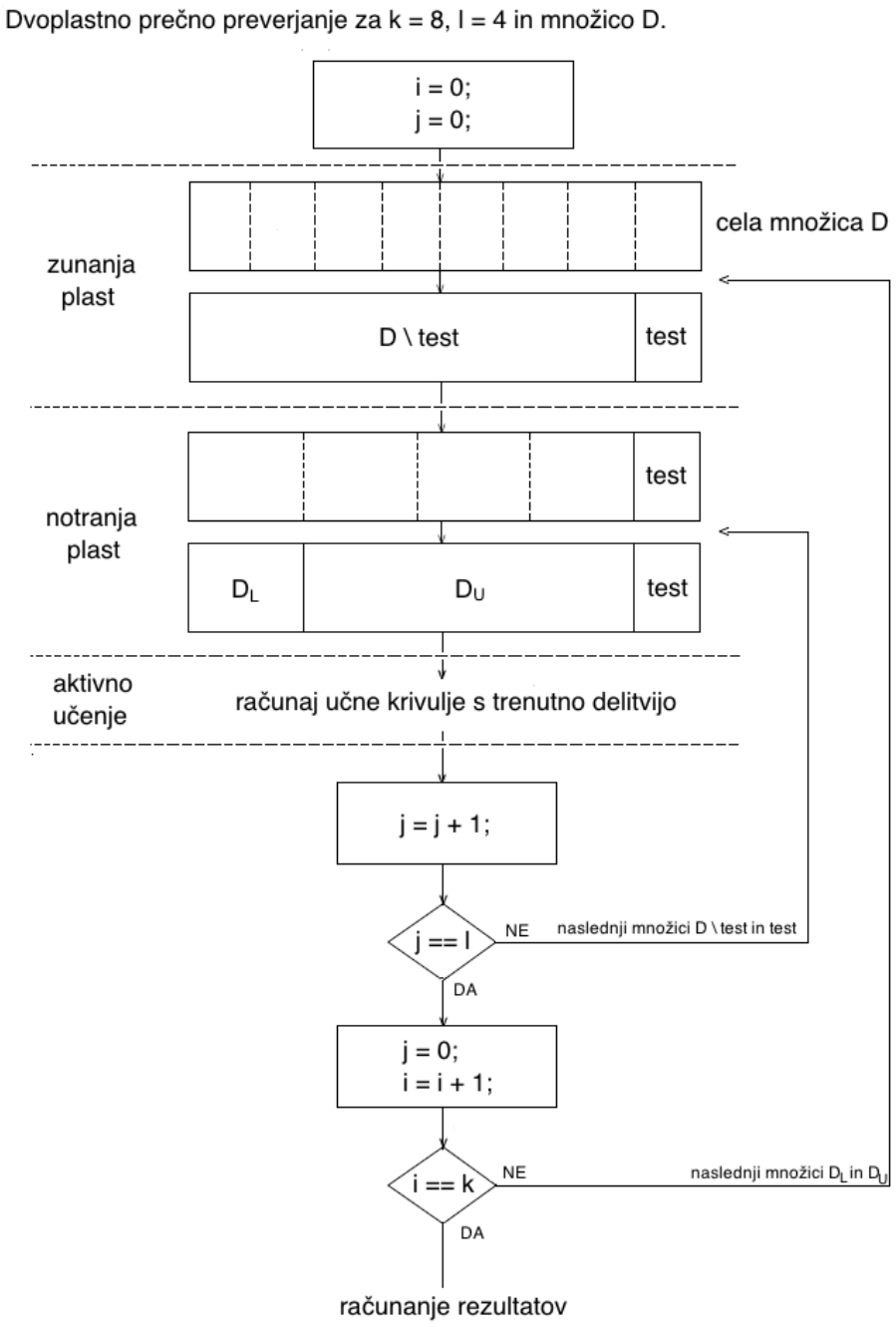
Takšna generalizacija je lahko tudi nenatančna, vendar pa se pogosto izkaže za dobro. Primer, ki ni relacija ImaZakonca, bi bila recimo relacija JePredsednikDržave. Dogodka z informacijo o kritičnem trenutku v času sta seveda to da je konkreten človek prevzel mesto predsednika v določeni državi ali pa, da ga je odstopil ali predal drugemu in zaključil svoje predsedstvo. Indikativen dogodek bi bil recimo, da je nekdo zmagal na volitvah, ali pa da je v skupini najbolj verjetnih zmagovalcev. Indikativen dogodek za konec veljavnosti relacije so recimo nove predsedniške volitve, kjer je omenjeno, da se predsedniku v obravnavi izteka čas. Na drugi strani pa je soroden dogodek katerikoli dogodek povezan s predsedništvom.

V diplomskem delu delimo dogodke v 7 skupin po zgornje opisanem principu: poroka (0), indikativno poroke (1), sorodno poroki (2), ločitev (3), indikativno ločitve (4), sorodno ločitvi (5) in nerelevantno (6), kjer vrednosti v oklepaju predstavljajo vrednosti ciljnih spremenljivk.

3.5 Simulacija in analiza sistema

V prejšnjem poglavju smo spoznali sistem, ki rešuje problem iskanja dogodkov z vsebovanim kritičnim trenutkom v času. V tem poglavju simuliramo in analiziramo sistem. Podsistemi inicializator, označevalec in aktivni učenec so precej enostavni, dočim pa to ne velja za podsistem analizator. Bolj podrobno si zato pogledamo princip določitve dobre konfiguracije modela učenja, kjer je največji poudarek na vplivu novo pridobljenih primeri na izboljšavo zgrajenega modela. V resnici je strategija aktivnega učenja za nas najbolj pomembna tudi zato, ker je označevanje novih primerov najdražje. Če bi značilno znižali število primerov, ki jih potrebujemo za izgradnjo dovolj dobrega modela, potem smo močno poenostavili edini del sistema, ki ga ni moč avtomatizirati. V vsakem primeru pa s tem dobimo vpogled v natančnost modela, ki ga lahko zgradimo iz le malo primerov za konkreten problem relacije.
Kot smo že omenili, za analizo sistema izberemo relacijo ImeZakonca. Tokrat odpravimo pomoč regularnih izrazov pri zmanjševanju začetne množice z namenom, da pridobljena množica bolje odraža realno stanje. Ne smemo pozabiti, da avtomatsko ni moč pridobiti množice predpon, ki nam je množico podatkov zmanjšala v začetnem poskusu, zato je problem, ki ne uporablja množice predpon dejansko uporaben v praksi in bolje generalizira problem na ostale relacije.

Množico podatkov ponovno pridobimo z poizvedbami po osebah iz začetnega poskusa. Tokrat je množica velika 2700 primerov, od katerih je 110 pozitivnih. Opazimo, da je sedaj množica še bolj neuravnotežena kot prej in je problem še težji. Primere označimo s podsistemom označevalec.



Slika 3.5: Slika ilustrira postopek dvoplastnega prečnega preverjanja, za konkretna parametra $k = 8$ in $l = 4$ in množico primerov D. V zunanji plasti se množica D deli na 8 delov, kjer določimo enega za testno množico. V notranji se D brez testne D *test* deli na 4 dele, in določimo množici D_L in D_U. Po tem simuliramo a iteracij aktivnega učenja in dobimo učno krivuljo. Nato priredimo množici D_L v različnih iteracijah vse kvadratke v notranji plasti in v vsaki iteraciji dobimo novo učno krivuljo. To storimo tudi v zunanji plasti, le da zdaj priredimo v vsaki iteraciji različen kvadratek testni množici.

Za analizo uporabimo dvoplastno prečno preverjanje. Takšno prečno preverjanje je sestavljeno iz dveh plasti, zunanje in notranje.

- V **zunanji plasti** celotno množico D razdelimo na testno množico $test$ in preostanek $D \setminus test$.

- V **notranji plasti** razdelimo množico $D \setminus test$ v označeno množico D_L in neoznačeno D_U. Vsi primeri so v resnici označeni, vendar se za namen simulacije aktivnega učenja pretvarjamo, da primeri v D_U niso označeni.

V obeh plasteh se notranje množice podatkov premešajo po principu k-kratnega prečnega preverjanja z namenom, da bolje ocenimo uspešnost modelov v praksi. Bralec si lahko ogleda ilustracijo postopka za sliki 3.5, v nadaljevanju pa podrobneje opišemo dogajanje v obeh plasteh prečnega preverjanja skozi celoten postopek evaluacije:

- Celotno množico podatkov delimo na učno D_L, testno $test$ in D_U. V a iteracijah vsakokrat učni množici dodamo m primerov iz D_U po nekem principu aktivnega učenja.

- Da izvemo, kako se modeli obnašajo na povprečnih učnih množicah s povprečnim D_U, na povprečni testni množici, delitev na množice opravimo s k-kratnim prečnim preverjanjem. To naredimo tako, da v zunanji plasti prečnega preverjanja celotno množico razdelimo na testno $test$ in preostanek $D \setminus test$. V notranji plasti končno delimo $D \setminus test$ na učno D_L in D_U. Zunanja plast se premeša k-krat, notranja plast pa l-krat.

- Poskus ponovimo za več različnih algoritmov in strategij aktivnega učenja, pri tem pa pazimo, da uporabljamo isto naključno seme, ki zagotavlja ponovljivost poskusa, oziroma isto delitev množic v vsakem poskusu. To je nujno, da je primerjava med algoritmi in strategijami poštena.

Zgornji postopek ima kar nekaj parametrov. k določa kolikokrat se premeša zunanja plast, l kolikokrat notranja plast, a določa koliko iteracij aktivnega učenja opravimo in m koliko primerov prenesemo iz D_U v D_L v vsaki iteraciji aktivnega učenja. Smiselno je, da je testna množica velika približno 10%, zato je smiseln parameter $k = 10$. Parametra a in m določata število točk na dobljeni učni krivulji. Ostane še vprašanje kakšen naj bo parameter l v notranji plasti prečnega preverjanja oziroma kako velika naj bo začetna učna množica D_L in kako velika D_U. Množica D_L je lahko tako velika, da se AUC precej ustali že v začetni iteraciji aktivnega učenja, pri čimer ne bi bilo mogoče oceniti njegovega vpliva. Zato mora biti množica D_L dovolj majhna. Z namenom izbire pravega razmerja med D_L in D_U izvedemo sledeč poskus. Želimo izvedeti kako narašča AUC z povečevanjem števila primerov. To naredimo tako, da zgradimo model z majhno množico primerov, potem pa jo postopoma povečujemo in gradimo nove modele. Pri tem opazujemo kako se obnaša AUC modelov v odvisnosti od količine primerov, t.j. gradimo učno krivuljo. Primere zaenkrat izbiramo naključno, saj naš namen ni primerjava aktivnega učenja, ampak samo izbira dobrega razmerja med D_U in D_L. Na dobljeni učni krivulji tako iščemo točko, kjer AUC ni premajhen, dočim pa se kasneje še vseeno znatno poveča. To dosežemo na dva načina:

1. Najprej vzamemo 600 primerov iz celotne množice za D_L in izračunamo AUC, potem pa 20-krat dodamo 100 primerov v D_L ter vsakič izračunamo AUC. AUC preverjamo tako, da množico delimo na učno (80%) in testno (20%). Ta postopek ponovimo 10-krat, vsakič z drugačno permutacijo celotne množice, da dobimo nek povprečen rezultat (*monte carlo* prečno preverjanje).

2. Iz celotne množice vzamemo približno 270 primerov, kar služi kot testna množica. Iz preostale množice jih izberemo 600 za učno množico, potem pa ji zopet 16-krat dodamo po 100. Pred vsakim dodajanjem izračunamo AUC ter celoten postopek tudi tu ponovimo desetkrat, vsakič z novo permutacijo celotne množice.

Razlika med prvim in drugim načinom je v tem, da je v drugem poskusu testna množica fiksna, v prvem poskusu pa se premeša. Vsak poskus ima prednost, prvi je bolj realen, saj gleda kako se model obnaša na povprečni testni množici, vendar bo zaradi majhnega števila primerov standardna deviacija precej visoka. Drugi primer je sicer pristranski, ker vedno testiramo na isti množici - lahko se zgodi, da je nek model dober zgolj na tej, vendar pa zaradi majhne deviacije dobimo boljši občutek za pravo razmerje med D_U in D_L v praksi. Rezultati obnašanja pri naključnem izbiranju kažejo na to, da je vpliv aktivnega učenja bolj izrazit takrat ko je primerov malo, zato se odločimo za $l = 5$. Večji del razdeljene množice pripada D_U, manjši pa D_L. Parametri poskusa so torej $k = 9$, $l = 5$, $a = 8$ in $m = 150$, kjer sta a in m nastavljena glede na računske zmožnosti. Rezultati poskusa so predstavljeni v poglavju 4.

3.6 Razločevanje kritičnih trenutkov v času

Problem, ki ga v diplomski nalogi rešujemo, smo naslovili že v prejšnjem poglavju. Da bi imela rešitev kar se da veliko praktične vrednosti, želimo za najdene dogodke poznati vse podrobnosti. Za najden dogodek, ki vsebuje kritični trenutek v času želimo vedeti:

- "Na katero relacijo se navezuje?"
- "Kdaj se je zgodil?"
- "Kakšen tip časovne točke vsebuje?"
- "Na katero instanco relacije se navezuje?"

Z iskanjem dogodkov s kritičnimi trenutki v času že implicitno dobimo odgovora na prvi dve vprašanji. Glede na to, da gradimo klasifikator C_r za konkretno relacijo r, ko najdemo dogodek s c_r vemo, da se navezuje na relacijo r. Dogodki so v sistemu EventRegistry opremljeni s časom dogodka, kar odgovarja na drugo vprašanje.

Odgovor na tretje vprašanje "Kakšen tip časovne točke vsebuje?"bi lahko naivno pridobili tako, da bi pogledali poizvedbe iz katere smo dogodek dobili. Če smo ga dobili s poizvedbo po začetni množici konceptov S, bi rekli, da vsebuje začetno časovno točko, če pa smo poizvedbo opravili s končno množico F pa bi rekli, da vsebuje končno časovno točko. Takšna hevristika se izkaže za slabo, saj se pri poizvedovanju po množici S v vrnjeni množici dogodkov znajdejo tudi dogodki, ki se v resnici navezujejo na množico F in obratno. To se zgodi zato, ker so koncepti v S in F pogosto zelo sorodni. Problema se zato raje lotimo s strojnim učenjem.

Uvrščanje dogodkov kritičnimi trenutki v času po tipu

Ker je množica dogodkov že označena in smo jo označili na zelo splošen način, lahko nek klasifikator naučimo uvrščati dogodke po tipih. Tako zgradimo en klasifikator za prvi tip in en klasifikator za drugi tip dogodkov. V naši nalogi za relacijo ImaZakonca(x,y) razpoznavamo različne kritične trenutke v času ločeno, na sledeč način:

- Razpoznavanje **začetnih** časovnih točk (poroke). Dogodke tipa poroka, indikativno poroke in sorodno poroki določimo za pozitiven razred, vse ostale pa za negativnega. Dogodke predstavimo kot primere, kjer atributni prostor sestavimo z metodo TFIDF nad povzetkom dogodkov, nato pa uvrščamo z algoritmom SVM.

- Razpoznavanje **končnih** časovnih točk (ločitve). Dogodke tipa ločitev, indikativno ločitve in sorodno ločitvi določimo za pozitiven razred, druge pa za negativnega. Uvrščanje poteka enako kot zgoraj.

V praksi bi za katerikoli dogodek najprej določili, če vsebuje referenčno časovno točko, s klasifikatorjem, ki išče takšne dogodke. Če bi se izkazalo, da gre za pozitiven primer, potem bi z naslednjim klasifikatorjem določili točen tip referenčne točke, ki jo dogodek vsebuje. Kako dobro se lahko s podatki, ki smo jih označili, naučimo razločevati med porokami in ločitvami predstavimo v naslednjem poglavju.

Poglavje 4

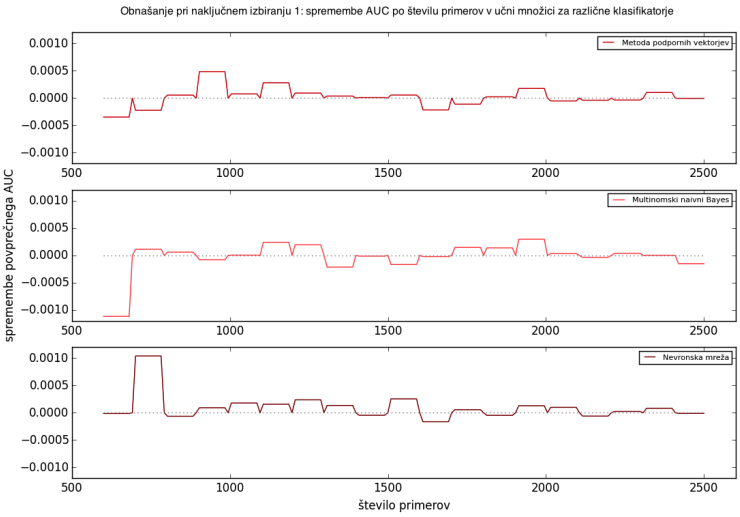
Rezultati

4.1 Simulacija in analiza sistema

4.1.1 Obnašanje pri naključnem izbiranju

Rezultati poskusov obnašanje pri naključnem izbiranju na slikah 4.1, 4.2, 4.3 in 4.4 kažejo, da se spremembe v učinkovitosti najbolj kažejo pri majhnih učnih množicah. Za bolj nazoren prikaz sprememb v povprečnem AUC glede na število primerov v učni množici, se odločimo prikazati spremembe AUC po številu primerov. To naredimo tudi zato, ker je naš namen preizkus vpliva števila primerov na spremembe povprečnega AUC in ne uspešnosti klasifikatorjev. Uspešnost klasifikatorjev obravnavamo v podpoglavju 4.1.2, kjer v obzir vzamemo ne le naključno izbiranje kot strategijo aktivnega učenja, temveč tudi druge.

Sliki 4.1, 4.2 prikazujeta rezultate za poskus naključnega izbiranja s spreminjajočo testno množico. Na sliki 4.1 levo je lepo razvidno, da se največ sprememb v povprečnem AUC zgodi pri majhnem številu primerov. Zdi se, da SVM najhitreje doseže končno vrednost AUC, dočim naivni Bayesov klasifikator in nevronska mreža pri majhnem številu primerov kažeta še precej skokovite spremembe. Na sliki 4.2 desno so standardne deviacije AUC za različno število primerov. Kot vidimo so le-te precej visoke, kar verje-



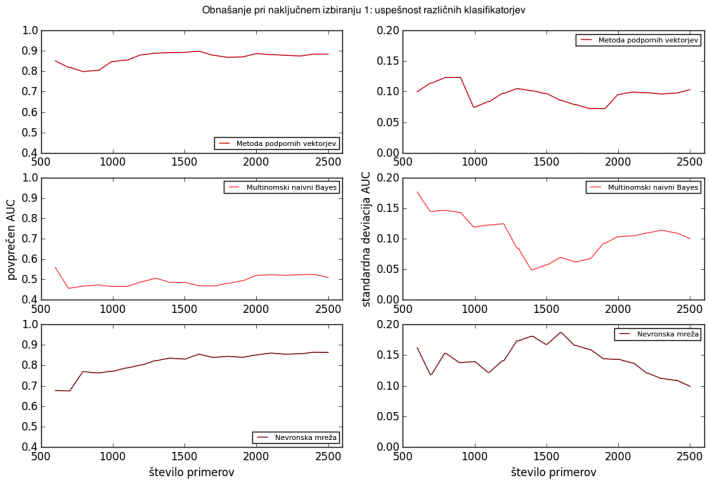
Slika 4.1: Slika prikazuje rezultate za naključno izbiranje primerov s spremenljivo testno množico, kjer primerjamo spremembe AUC po številu primerov v učni množici za različne klasifikatorje.

tno pomeni, da nimamo dovolj podatkov, da bi lahko zanesljivo izračunali kako učinkoviti so naučeni klasifikatorji. Na sliki 4.2 levo pa vidimo kakšno uspešnost so dosegali klasifikatorji, ki je precej visoka.

Sliki 4.3, 4.4 prikazujeta rezultate za poskus naključnega izbiranja s fiksno testno množico. Krivulje na sliki 4.3 so precej bolj umirjene kot na sliki 4.1. Izjema je nevronska mreža, kjer so spremembe vidne dokler ima učna množica manj kot 1500 primerov. SVM se zdi najbolj zanesljiv, kar bi pomenilo tudi, da zelo hitro razbere vse potrebne informacije za dobro delovanje medtem, ko naivni Bayesov klasifikator in nevronska mreža še zmeraj pridobivata nove informacije za boljše delovanje.

4.1.2 Glavni del

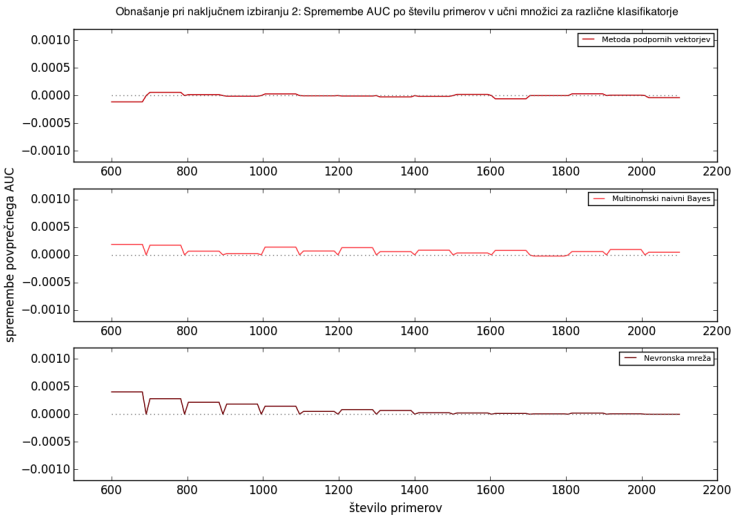
V nadaljevanju predstavimo rezultate glavnega dela diplomskega dela. Preverimo kako dobre in zanesljive klasifikatorje je moč zgraditi s podatki, ki so



Slika 4.2: Slika prikazuje rezultate za naključno izbiranje primerov s spremenljivo testno množico, kjer primerjamo uspešnost različnih klasifikatorjev. Na levi je prikazan povprečen AUC v odvisnosti od števila primerov, na desni pa standardna deviacija AUC.

nam na voljo ter kako strategije aktivnega učenja vplivajo na učenje.

Za učenje izberemo iste tri algoritme kot v prejšnjih poskusih, primerjamo pa naslednje strategije aktivnega učenja: Metodo naključnega izbiranja, metodo najmanjšega zaupanja, metodo najmanjšega roba, metodo najmanjšega zaupanja s kosinusno korelacijo, ki so vse definirane v poglavju 2.7. Preizkusimo še eno novo strategijo aktivnega učenja, imenovano metoda najmanjše razdalje od povprečne verjetnosti. Ta je plod lastnega razmišljanja in pravi, da so najbolj informativni primeri v naoznačeni množici tisti, katerih izhodne vrednosti klasifikatorja so najbližje povprečni izhodni vrednosti klasifikatorja v neoznačeni množici. Rečemo ji metoda najmanjše razdalje od povprečne verjetnosti zato, ker so vsi izhodi klasifikatorjev v diplomskem delu preslikani v verjetnosti prostor. Definicija se je zdela smiselna zaradi neuravnoteženosti porazdelitve razredov, katere posledica so tudi neuravnoteženi izhodi klasifikatorjev. Primer tega je, da ima negativen primer izhodno verjetnost $p = 0.97$ za to, da je primer pozitiven, še zmeraj pa ga lahko pravilno klasificiramo

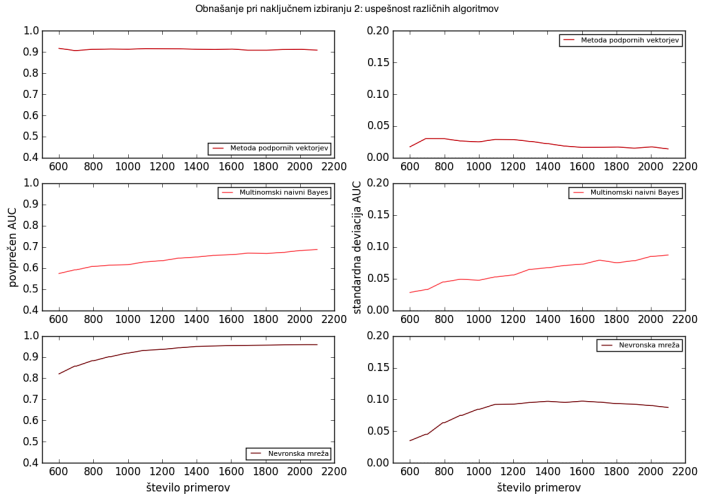


Slika 4.3: Slika prikazuje rezultate za naključno izbiranje primerov s fiksno testno množico, kjer primerjamo spremembe AUC po številu primerov v učni množici za različne klasifikatorje.

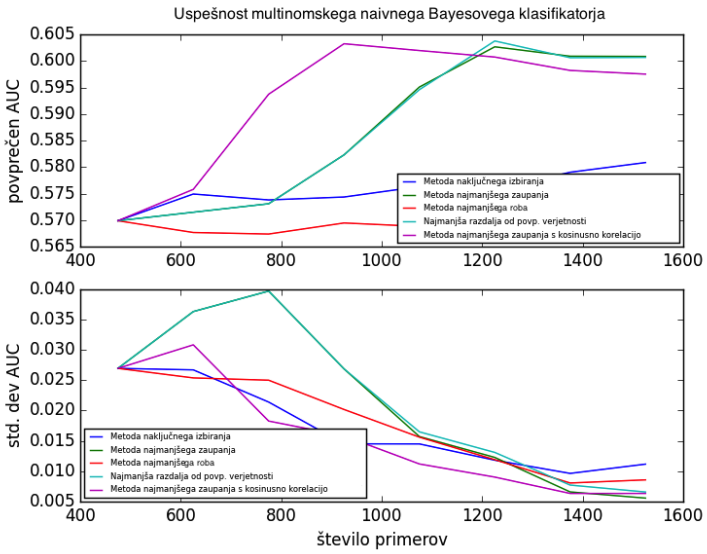
kot negativnega, ker so verjetnosti za pozitivne še večje, torej je tudi tako visoka verjetnost v resnici majhna zaradi neuravnoteženosti problema.

Poskuse za vse algoritme izvajamo na označeni množici 2700 primerov, z dvostopenjskim prečnim preverjanjem kot je prikazano v poglavju 3.5, kjer je testna množica velika $\frac{1}{9}$ primerov oziroma okoli 300, začetna učna množica je velika $\frac{8}{9}\frac{1}{5} = \frac{8}{45}$ primerov oziroma 475, ostalo pa je neoznačena množica namenjena aktivnemu učenju. Prečno preverjanje premeša množice 45 krat in vsakič izračuna učno krivuljo za AUC v osmih iteracijah aktivnega učenja, v vsaki od slednjih pa učni množici doda 150 primerov iz neoznačene množice po različnih strategijah aktivnega učenja. Na koncu smo te krivulje povprečili po vseh iteracijah prečnega preverjanja in izračunali standardne deviacije. Te povprečne krivulje, ter njihove standardne devacije so prikazane na slikah 4.5, 4.6, 4.7, 4.8, v nadaljevanju pa jih diskutiramo.

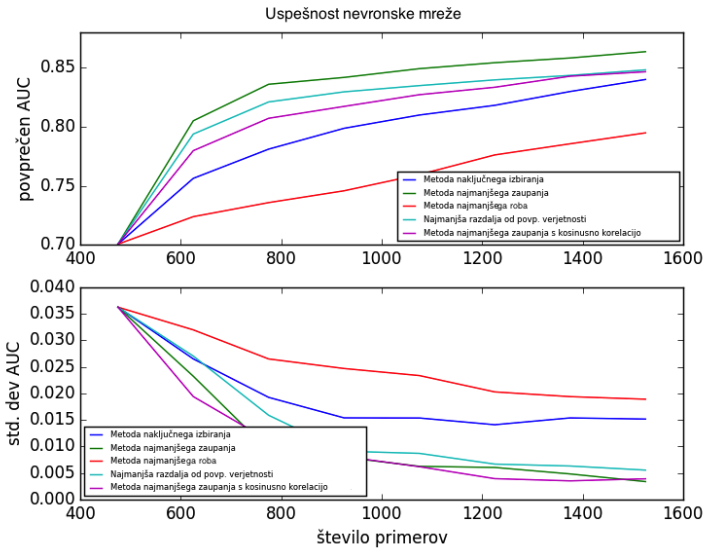
Slika 4.5 prikazuje uspešnost naivnega Bayesovega klasifikatorja. Sam klasifikator ne dosega zadovoljive uspešnosti, še zmeraj pa opazimo, da mu



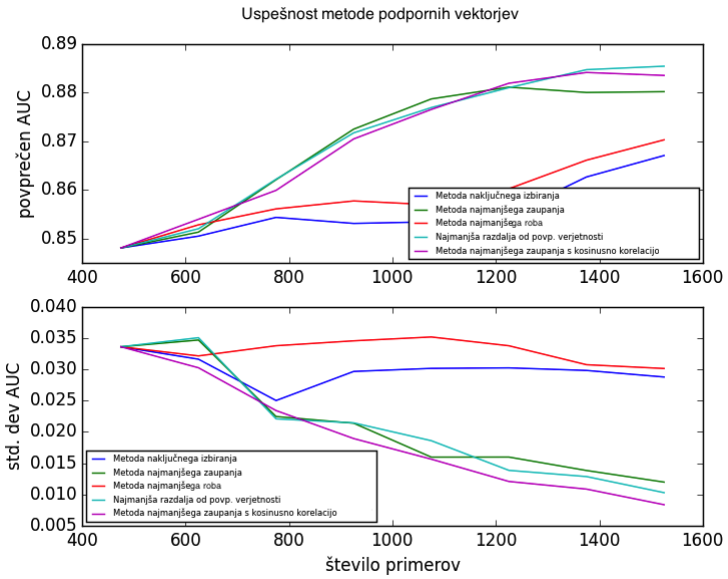
Slika 4.4: Slika prikazuje rezultate za naključno izbiranje primerov s spremenljivo testno množico, kjer primerjamo uspešnost različnih klasifikatorjev. Na levi je prikazan povprečen AUC v odvisnosti od števila primerov, na desni pa standardna deviacija AUC.



Slika 4.5: Slika prikazuje uspešnost naivnega Bayesovega klasifikatorja pri različnih strategijah aktivnega učenja.



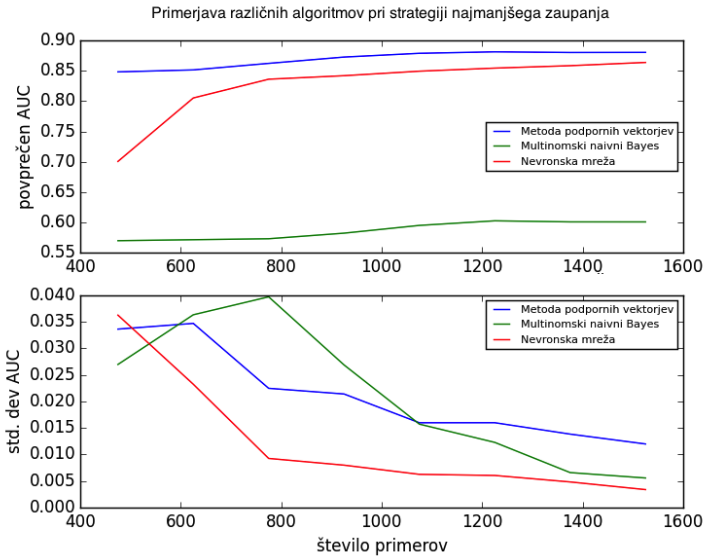
Slika 4.6: Slika prikazuje uspešnost nevronske mreže pri različnih strategijah aktivnega učenja.



Slika 4.7: Slika prikazuje uspešnost klasifikatorja SVM pri različnih strategijah aktivnega učenja.

aktivno učenje pomaga izbrati informativne primere za učenje. Na sliki 4.5 zgoraj je prikazana učna krivulja povprečnega AUC, spodaj pa je prikazana standardna deviacija AUC. Zgoraj vidimo, da je na začetku najboljša strategija aktivnega učenja metoda najmanjšega zaupanja s kosinusno korelacijo. Očitno je upoštevanje korelacij temu klasifikatorju precej pomagalo. Strategije, kjer korelacij ne upoštevamo, še zmeraj dosegajo boljše rezultate kot naključno izbiranje. Izjema je metoda najmanjšega roba, ki je slabša od naključnega izbiranja. Možen razlog je neuravnoteženost porazdelitve razredov in posledično neuravnotežene izhodne vrednosti klasifikatorjev.

Slika 4.6 prikazuje uspešnost nevronske mreže pri različnih strategijah aktivnega učenja. Prav tako je zgoraj prikazana učna krivulja povprečnega AUC in spodaj standardna deviacija AUC. Kot vidimo na sliki 4.6 zgoraj, dosega nevronska mreža precej boljše rezultate kot naivni Bayesov klasifikator (slika 4.5). Strategije aktivnega učenja pomagajo pri izbiri informativnih primerov. Presenetljivo metoda kjer upoštevamo korelacijo med primeri ne



Slika 4.8: Slika prikazuje primerjavo uspešnosti različnih klasifikatorjev pri metodi najmanjšega zaupanja kot strategiji aktivnega učenja, ki se je izkazala za najboljšo.

prinaša najboljših rezultatov. Morda so primeri tako različni ali čudno nagručeni, da jim upoštevanje korelacij ne pomaga. Lahko bi poskusili gručiti primere v več gruč in korelacijo meriti zgolj znotraj teh gruč.

Slika 4.7 prikazuje uspešnost SVM pri različnih strategijah aktivnega učenja. Tudi tukaj je zgoraj prikazana učna krivulja povprečnega AUC in spodaj standardna deviacija AUC. Kot je vidno na sliki 4.7 zgoraj, dosega SVM najboljše rezultate, ki se približajo $AUC = 0.9$. Najbolj vplivna strategija aktivnega učenja je metoda najmanjše razdalje od povprečne verjetnosti. To bi lahko potrdilo našo predpostavko o temu, da je pomembno upoštevati tudi neuravnotežene izhodne verjetnosti zaradi neuravnotežene porazdelitve razredov. Tudi algoritmu SVM aktivno učenje z upoštevanjem korelacij ne pomaga toliko kot druge strategije. Razlog za to je verjetno isti kot za nevronske mreže. Poskusiti bi morali z gručenjem primerov in računanjem korelacije znotraj gruč. Razlike v uspešnosti algoritma SVM so precej majhne. Izgleda, da temu algoritmu zadostuje precej majhno število primerov za učenje dobre hipoteze. Zanimivo je, da SVM hitro doseže maksimalno uspešnost in se kasneje preneha znatno izboljševati. Mogoče je že iz začetnih primerov pridobil večino informacij, ki jih ima množica podatkov na voljo, z novimi primeri pa je večinoma prejemal redundantne informacije.

Na sliki 4.8 primerjamo različne algoritme pri metodi najmanjšega zaupanja kot strategiji aktivnega učenja. Na zgornji strani sliki 4.8 primerjamo povprečen AUC, na spodnji pa njegovo standardno deviacijo. V povprečju najvišji povprečen AUC dosega SVM, ki mu sledi nevronska mreža in daleč za njima multinomski naivni Bayesov klasifikator. SVM doseže le malo višji povprečni AUC kot nevronska mreža, dočim pa ima nevronska mreža nižjo standardno deviacijo. To pomeni, da moramo s povprečnim AUC prikazano učinkovitost teh klasifikatorjev vzeti z rezervo, saj je verjetno veliko izračunanih klasifikatorjev z nevronsko mrežo dosegalo boljšo uspešnost kot nekateri SVM modeli. Primer tega sicer vidimo tudi na sliki 4.4, kjer nevronska mreža doseže znatno višji AUC kot SVM, poudarimo pa, da je eksperiment zastavljen s fiksno testno množico. Na sliki 4.8 se vidi tudi na nevronska

mreža še zmeraj raste v povprečnem AUC tudi, ko je primerov več kot 1500, dočim SVM takrat raste precej počasneje. To bi lahko pomenilo tudi, da bi NN presegel SVM v povprečnem AUC, če bi imeli več učnih primerov. S tem znanjem ugotovimo, da bi v resnici potrebovali več primerov, da bi lahko zanesljivo določili kateri klasifikator je najboljši. Rezultate glede izbire algoritma zato vzamemo z rezervo.

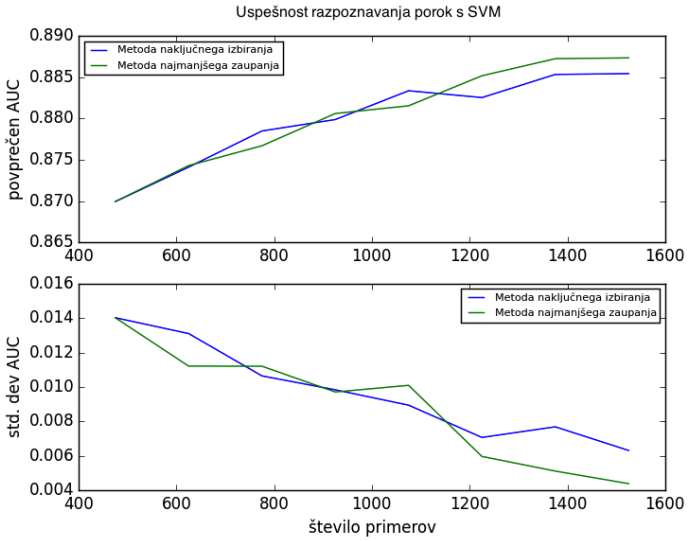
Naivni Bayesov klasifikator je za učenje daleč najhitrejši klasifikator, kar ga naredi zanimivega za obravnavo. Žal očitno atributi v prostoru atributov niso dovolj neodvisni, da bi lahko naivni Bayesov klasifikator dosegal dobre rezultate. Če bi povprečen AUC rastel pri večji količini primerov bi lahko sklepali, da se bo eventualno naučil dobrega klasifikatorja, vendar vidimo na sliki 4.8 zgoraj, da se temu klasifikatorju povprečen AUC preneha zviševati po 1200 primerih, kar bi pomenilo, da tudi pri veliko večji učni množici ne bo generiral dobrih modelov. To je škoda, ker sta SVM in nevronska mreža mnogo počasnejša, kar pomeni, da bosta pri velikih količinah primerov potrebovala zelo dolgo časa za učenje. V praksi bi zato verjetno uporabljali inkrementalne različice teh algoritmov.

4.2 Poroke in ločitve

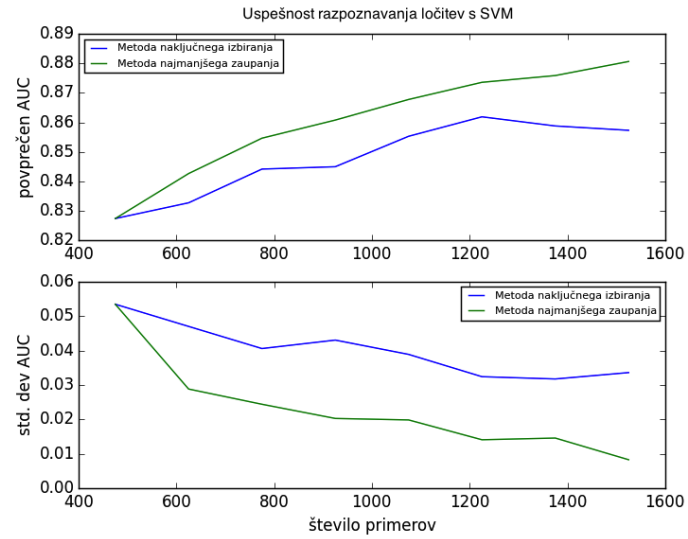
Na slikah 4.9 in 4.10 predstavljamo tudi rezultate za uspešnost uvrščanja dogodkov med tiste, ki so povezani s poroko in tiste, ki so povezani z ločitvijo, z namenom ugotavljanja tipa kritičnega trenutka v času, vsebovanega v dogodku. Na slikah 4.9 in 4.10 zgoraj so predstavljene učne krivuje povprečnega AUC, spodaj pa standardne deviacije. Uporabimo algoritem SVM in primerjamo strategiji aktivnga učenja naključno izbiranje in najmanjša zanesljivost.

Problem smo zastavili kot dva binarna klasifikacijska problema, kjer slika 4.9 predstavlja uspešnost razpoznavanja dogodkov, ki se navezujejo na poroke, slika 4.10 pa uspešnost razpoznavanja dogodkov, ki se navezujejo na ločitve.

Poroke smo razpoznavali tako, da smo dogodke, ki spadajo v skupine po-



Slika 4.9: Slika prikazuje uspešnost uvrščanja dogodkov v dogodke o porokah z metodo SVM in dveh strategijah aktivnega učenja.



Slika 4.10: Slika prikazuje uspešnost uvrščanja dogodkov v dogodke o ločitvah z metodo SVM in dveh strategijah aktivnega učenja.

rok, indikacij porok in sorodnih porokam določili za pozitivne, ostale pa za negativne. Takšnih dogodkov je v celotni množici podatkov 573, ostalih pa okoli 2100. Slednje smo določili za negativne. Izkaže se, da je razpoznavanje porok približno enako težek problem kot glavni problem (prikazan na sliki 4.8), saj dosegamo s SVM zelo podobne rezultate. Zanimivo je, da so rezultati podobni, saj je ta problem precej bolj uravnotežen kot v porazdelitvi razredov kot glavni problem. Spomnimo, da je bilo v glavnem problemu le 110 pozitivnih.

Ločitve smo razpoznavali tako, da smo za pozitivne določili dogodke v skupinah ločitve, indikativni za ločitve in sorodni ločitvam. Takšnih je bilo zgolj 123, kar je po uravnoteženosti podobno glavnemu problemu. Imamo torej 123 pozitivinih primerov in približno 2580 negativnih. Rezultati prikazani na sliki 4.10 kažejo, da dosegamo rahko nižji povprečni AUC kot pri razpoznavanju porok, za kar bi lahko bil razlog tudi manjše število pozitivnih primerov. Tudi standardna deviacija AUC je višja kot pri razpoznavanju porok, kar indicira, da klasifikatorjih na različnih testnih množicah dosegajo zelo različne povprečne AUC. Pri obeh problemih sicer dosegamo visok povprečen AUC blizu 0.9. Zanimivo pa je videti, da je dogodkov povezanih s porokami veliko več kot tistih, ki so povezani z ločitvami. Iz tega lahko sklepamo, da bo tudi pri drugih časovnih relacijah problem to, da bomo lažje razpoznavali en tip kritičnih trenutkov v času kot drug.

Poglavje 5

Sklep

V diplomskem delu smo spoznali in preučili problem časovnih relacij v sistemu NELL in predlagali rešitev za problem iskanja dogodkov, ki vsebujejo kritične trenutke v času za časovne relacije, s pomočjo sistema, ki nam te dogodke ponuja. Razvili smo sistem, ki se uči in komunicira s spletno stranjo, kjer eksperti označujejo podatke za učenje, ki se označeni pošiljajo nazaj sistemu in izkoristijo za nadaljne učenje klasifikatorjev. Ti podatki so izbrani s strategijo aktivnega učenja zavoljo tega, da bi jih bilo potrebno označiti čim manj, da bi klasifikatorji problem reševali dovolj dobro za uporabo v praksi. Sistem smo simulirali in analizirali na konkretni relaciji ImaZakonca(x,y), kjer se je izkazal za zanesljivo rešitev. Ker je to le del reševanja večjega problema časovnih relacij, kjer si želimo v dogodkih z kritičnim trenutkom v času razpoznati različne prodrobnosti (za katero relacijo gre, kdaj se je dogodek zgodil, kakšen tip kritičnega trenutka v času vsebuje, katera je relevantna instanca relacije), smo zavoljo tega, obravnavali tudi problem, kako pridobiti tip kritičnega trenutka v času, ki ga dogodek vsebuje. Za ta namen smo specializirali dogodke v več kategorij, ki so omogočile, da smo problem rešili zanesljivo.

Prihodnje delo, ki bi dodalo precej več praktične vrednosti rešitvi, bi bilo iskanje instance na katero se navezuje dogodek. Verjetno bi uporabili informacijo o konceptih, ki se najpogosteje pojavljajo v člankih, navezujočih se

na dogodek v vprašanju, vendar le-to verjetno ne bi bilo dovolj za zanesljivo rešitev. Za problem iskanja dogodkov s kritičnim trenutkom v času bi lahko preizkusili še številne druge metode strojnega učenja, ki bi morda dosegle še boljše rezultate od teh. Možno izboljšavo predstavlja tudi ideja, da klasificiramo dogodke z regresijo, kjer za neko časovno relacijo dejanskim dogodkom z informacijo o kritičnih trenutkih v času nastavimo ciljno spremenljivko na 3, indikativnim dogodkom na 2, sorodnim na 1 in irelevantnim na 0. Mogoče bi tako lahko našli prag za dobro napoved v testni množici, saj bi upoštevali tudi dejstvo, da so si dogodki, ki jih ne iščemo, med seboj različni. Morda bi lahko z označenimi podatki poskušali razširiti referenčni množici S in F za različne relacije, z iskanjem najbolj pogostih glagolov, ki se pojavljajo v besedilih dogodkov.

Poglavje 6

Dodatki

6.1 Programska koda

Programske kode je veliko in bi v svoji celoti močno povečala dolžino tega dokumenta, zato smo se odločili da vljučimo le en zanimiv del kode. Gre za glavno zanko analize in simulacije glavnega sistema. Sicer implementacije notranjih funkcij ne bomo podali zaradi dolžine dokumenta, je pa že iz imen funkcij jasno kaj neka funkcija naredi.

```
# iterate through each algorithm, AL strategy pair
for alg in algs:
    for al_str in al_strs:

        # dont do the same work twice!
        if results[al_str][alg][0][0][0] != 0:
                continue

        print(al_str + '_' + alg)
        all_events = dtools.SQLiteEventAccessor('data/main.db').get_labeled_all()

        # shuffle and ensure the same random seed for all alg,alstr pairs
        random.seed(rnd_seed)
        random.shuffle(all_events)

        # determine the experiment by label transform and init active learner
        vectorizer, D_L = mtools.evts_to_ftr_mat(all_events, ['summary'])
        labels = [e['label'] for e in all_events]
        labels = mtools.transform_labels_experiment1(labels)
        active_learner = mtools.ActiveLearner(D_L, rstate=rnd_seed)

        # initialize outer cross validation
        outer_i = 0
        skf = StratifiedKFold(labels, n_folds=outerskf,random_state=rnd_seed)
        # A_0 are indices training set, B_0 indices in testing set
```

```
        for A_0, B_0 in skf:
            # build the test set
            print('outer_cv:_' + str(outer_i))
            test = mtools.build_csr_matrix([D_L[B_0[i]] for i in range(len(B_0))])
            test_labels = np.array([labels[B_0[i]] for i in range(len(B_0))])

            inner_i = 0
            inner_skf = StratifiedKFold([labels[i] for i in A_0], n_folds=innerskf,
                 random_state=rnd_seed)

            # inner cross validation
            # B_1 indices in D_U, A_1 indices in current training set
            for B_1, A_1 in inner_skf:

                # form the indices such that they index into the main data structure
                D_U = set([A_0[B_1[i]]for i in range(len(B_1))]) # for the sake of
                     performance
                train_indices = [A_0[A_1[i]] for i in range(len(A_1))]

                lc = [] # buffer for current iteration learning curve
                # iterate through active learning iterations
                for aliter in range(al_iter_n):
                    print('_iter' + str(aliter) + '_' + 'train:'+str(len(train_indices))
                         )
                    # build the training set
                    train = mtools.build_csr_matrix([D_L[train_indices[i]] for i in
                         range(len(train_indices))])
                    train_labels = np.array([labels[ train_indices[i] ] for i in range(
                         len(train_indices))])

                    auc, clf = mtools.Classifier(alg,rstate=rnd_seed).get_auc_ret_clf(
                         train, train_labels, test, test_labels)
                    lc.append(auc)

                    #selected = active_learner.uncertainty(list(D_U),clf,method=al_str,
                         n=al_periter)
                    selected = active_learner.correlation(list(D_U), clf, umethod='
                         least_confidence', beta=1, n=al_periter)
                    train_indices = np.append(train_indices, selected)
                    for item in selected:
                        D_U.remove(item)
                    pass
                # add the current iteration's learning curve
                results[al_str][alg][outer_i][inner_i] = lc
                inner_i = inner_i + 1
            outer_i = outer_i + 1
        # persist the results
        pickle.dump(results, open(results_file, 'wb'))
pass
```

Literatura

[1] I. Kononenko in M. R. Šikonja. *Inteligentni sistemi.* Založba FE in FRI, 2010.

[2] S. Thrun in T. Mitchell. *Lifelong Robot Learning.* Springer Berlin Heidelberg, 1995.

[3] S. Thrun in L. Pratt. *Learning to Learn.* Stringer Science & Business Media, 2012.

[4] J. E. Laird, N. Allen in P. S. Rosenbloom, "Soar: An architecture for general intelligence", *Artificial Intelligence*, št. 33, zv. 1, str. 1–64, 1987.

[5] P. Langley, K. B. McKusick, J. A. Allen, W. F. Iba in K. Thompson, "A design for the ICARUS architecture", *ACM SIGART Bulletin*, št. 2, zv. 4, str. 104–109, 1991.

[6] M. Veloso, J. G. Carbonell, A. Perez, D. Borrajo in E. Fink, "Integrating planning and learning: The PRODIGY architecture", *Journal of Experimental & Theoretical Artificial Intelligence*, št. 7, zv. 1, str. 81–120, 1995.

[7] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson idr., "Never-ending learning", V: *Proceedings of the Conference on Artificial Intelligence (AAAI)*, str. 2302–2310, 2015.

[8] G. Leban, B. Fortuna, J. Brank in M. Grobelnik, "Event Registry: Learning about world events from news", V: *Proceedings of the International World Wide Web Conference (WWW)*, str. 107–110, 2014.

[9] G. Leban, B. Fortuna, J. Brank in M. Grobelnik, "Cross-lingual detection of world events from news articles", V: *Proceedings of the International Semantic Web Conference (ISWC)*, str. 21–24, 2014.

[10] I. Kononenko. *Strojno učenje.* Založba FE in FRI, 1997.

[11] I.H. Witten in F. Eibe. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2005.

[12] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms", *Pattern Recognition*, št. 30, zv. 7, str. 1145–1159, 1997.

[13] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection", V: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, str. 1137–1145, 1995.

[14] F. N. Patel in N. R. Soni, "Text mining: A brief survey", *International Journal of Advanced Computer Research*, št. 2, zv. 4, str. 243–248, 2012.

[15] F. Sebastiani, "Machine learning in automated text categorization", *ACM Computing Surveys*, št. 34, zv. 1, str. 1–47, 2002.

[16] J. Leskovec, A. Rajaraman in J. D. Ullman. *Mining of Massive Datasets.* Cambridge University Press, 2014.

[17] Y. Fu, X. Zhu in B. Li, "A survey on instance selection for active learning", *Knowledge and Information Systems*, št. 35, zv. 2, str. 249–283, 2012.

[18] N. Cristianini in J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.* Cambridge university press, 2000.

[19] Implementacija naivnega Bayesovega klasifikatorja, scikit-learn. [Online]. Dosegljivo: `http://scikit-learn.org/stable/modules/naive_bayes.html`. [Dostopano 1.9.2015].

[20] Implementacija večnivojskega perceptrona, scikit-neuralnetwork. [Online]. Dosegljivo: `http://scikit-neuralnetwork.readthedocs.org/en/latest/module_mlp.html`. [Dostopano 1.9.2015].

[21] Implementacija metode podpornih vektorjev, scikit-learn. [Online]. Dosegljivo: `http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`. [Dostopano 1.9.2015].

[22] Implementacija stratificiranega prečnega preverjanja, scikit-learn. [Online]. Dosegljivo: `http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedKFold.html`. [Dostopano 1.9.2015].